Foundry is a smart contract development toolchain.

Foundry manages your dependencies, compiles your project, runs tests, deploys, and lets you interact with the chain from the command-line and via Solidity scripts.

📖 **Contributing**

You can contribute to this book on GitHub.

Sections

Getting Started

To get started with Foundry, install Foundry and set up your first project.

Projects

This section will give you an overview of how to create and work with existing projects.

Forge Overview

The overview will give you all you need to know about how to use `forge` to develop, test, and deploy smart contracts.

Cast Overview

Learn how to use `cast` to interact with smart contracts, send transactions, and get chain data from the command-line.

Anvil Overview

Learn about `anvil`, Foundry's local node.

Chisel Overview

Learn how to use `chisel`, Foundry's integrated Solidity REPL.

Configuration

Guides on configuring Foundry.

- Configuring with `foundry.toml`
- Continuous Integration
- Integrating with VSCode
- Shell Autocompletion
- Static Analyzers
- Integrating with Hardhat

Tutorials

Tutorials on building smart contracts with Foundry.

- Creating an NFT with Solmate
- Docker and Foundry
- Testing EIP-712 Signatures
- Solidity Scripting
- Forking Mainnet with Cast and Anvil
- Learning Foundry Videos

Contributing

Help us improve Foundry: Contributing

Appendix

References, troubleshooting, and more.

- FAQ
- CLI Reference
- forge Commands
- cast Commands
- anvil Commands
- chisel Commands
- Config Reference
- Cheatcodes Reference
- Forge Standard Library Reference
- DSTest Reference
- Miscellaneous

You can also check out Awesome Foundry, a curated list of awesome Foundry resources, tutorials, tools, and libraries!

Installation

If you face any issues while installing, check out the FAQ.

Precompiled binaries

Precompiled binaries are available from the GitHub releases page. These are better managed by using Foundryup.

Using Foundryup

Foundryup is the Foundry toolchain installer. You can find more about it here.

Open your terminal and run the following command:

```
curl -L https://foundry.paradigm.xyz | bash
```

This will install Foundryup, then simply follow the instructions on-screen, which will make the `foundryup` command available in your CLI.

Running `foundryup` by itself will install the latest (nightly) precompiled binaries: `forge`, `cast`, `anvil`, and `chisel`. See `foundryup --help` for more options, like installing from a specific version or commit.

> ℹ️ **Note**
>
> If you're on Windows, you will need to install and use Git BASH or WSL, as your terminal, since Foundryup currently does not support Powershell or Cmd.

Building from source

Prerequisites

You will need the Rust compiler and Cargo, the Rust package manager. The easiest way to install both is with `rustup.rs`.

Foundry generally only supports building on the latest stable Rust version. If you have an older Rust version, you can update with `rustup`:

```
rustup update stable
```

On Windows, you will also need a recent version of Visual Studio, installed with the "Desktop Development With C++" Workloads option.

Building

You can either use the different Foundryup flags:

```
foundryup --branch master
foundryup --path path/to/foundry
```

Or, by using a single Cargo command:

```
cargo install --git https://github.com/foundry-rs/foundry --profile local --
locked forge cast chisel anvil
```

Or, by manually building from a local copy of the Foundry repository:

```
# clone the repository
git clone https://github.com/foundry-rs/foundry.git
cd foundry
# install Forge
cargo install --path ./crates/forge --profile local --force --locked
# install Cast
cargo install --path ./crates/cast --profile local --force --locked
# install Anvil
cargo install --path ./crates/anvil --profile local --force --locked
# install Chisel
cargo install --path ./crates/chisel --profile local --force --locked
```

Installing for CI in Github Action

See the foundry-rs/foundry-toolchain GitHub Action.

Using Foundry with Docker

Foundry can also be used entirely within a Docker container. If you don't have it, Docker can be installed directly from Docker's website.

Once installed, you can download the latest release by running:

```
docker pull ghcr.io/foundry-rs/foundry:latest
```

It is also possible to build the docker image locally. From the Foundry repository, run:

```
docker build -t foundry .
```

For examples and guides on using this image, see the Docker tutorial section.

> ℹ️ **Note**
>
> Some machines (including those with M1 chips) may be unable to build the docker image locally. This is a known issue.

First Steps with Foundry

This section provides an overview of the `forge` command line tool. We demonstrate how to create a new project, compile, and test it.

To start a new project with Foundry, use `forge init`:

```
$ forge init hello_foundry
```

Let's check out what `forge` generated for us:

```
$ cd hello_foundry
$ tree . -d -L 1
.
├── lib
├── script
├── src
└── test

4 directories
```

We can build the project with `forge build`:

```
$ forge build
Compiling 27 files with 0.8.19
Solc 0.8.19 finished in 1.27s
Compiler run successful!
```

And run the tests with `forge test`:

```
$ forge test
No files changed, compilation skipped

Ran 2 tests for test/Counter.t.sol:CounterTest
[PASS] testFuzz_SetNumber(uint256) (runs: 256, μ: 30688, ~: 31310)
[PASS] test_Increment() (gas: 31325)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 14.43ms (14.04ms
CPU time)

Ran 1 test suite in 15.05ms (14.43ms CPU time): 2 tests passed, 0 failed, 0
skipped (2 total tests)
```

💡 **Tip**

> You can always print help for any subcommand (or their subcommands) by adding `--help` at the end.

You can watch these beginner tutorials if you are a visual learner.

Creating a New Project

To start a new project with Foundry, use `forge init`:

```
$ forge init hello_foundry
```

This creates a new directory `hello_foundry` from the default template. This also initializes a new `git` repository.

If you want to create a new project using a different template, you would pass the `--template` flag, like so:

```
$ forge init --template https://github.com/foundry-rs/forge-template
hello_template
```

For now, let's check what the default template looks like:

```
$ cd hello_foundry
$ tree . -d -L 1
.
├── lib
├── script
├── src
└── test

4 directories
```

The default template comes with one dependency installed: Forge Standard Library. This is the preferred testing library used for Foundry projects. Additionally, the template also comes with an empty starter contract and a simple test.

Let's build the project:

```
$ forge build
Compiling 27 files with 0.8.19
Solc 0.8.19 finished in 1.27s
Compiler run successful!
```

And run the tests:

```
$ forge test
No files changed, compilation skipped

Ran 2 tests for test/Counter.t.sol:CounterTest
[PASS] testFuzz_SetNumber(uint256) (runs: 256, μ: 30688, ~: 31310)
[PASS] test_Increment() (gas: 31325)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 14.43ms (14.04ms
CPU time)

Ran 1 test suite in 15.05ms (14.43ms CPU time): 2 tests passed, 0 failed, 0
skipped (2 total tests)
```

You'll notice that two new directories have popped up: `out` and `cache`.

The `out` directory contains your contract artifact, such as the ABI, while the `cache` is used by `forge` to only recompile what is necessary.

Working on an Existing Project

Foundry makes developing with existing projects have no overhead.

For this example, we will use PaulRBerg's `foundry-template`.

First, clone the project and run `forge install` inside the project directory.

```
$ git clone https://github.com/PaulRBerg/foundry-template
$ cd foundry-template
$ forge install
```

We run `forge install` to install the submodule dependencies that are in the project.

To build, use `forge build`:

```
$ forge build
Compiling 23 files with 0.8.23
Solc 0.8.23 finished in 1.29s
Compiler run successful!
```

And to test, use `forge test`:

```
$ forge test
No files changed, compilation skipped

Ran 3 tests for test/Foo.t.sol:FooTest
[PASS] testFork_Example() (gas: 3759)
[PASS] testFuzz_Example(uint256) (runs: 1000, μ: 8402, ~: 8402)
[PASS] test_Example() (gas: 8676)
Suite result: ok. 3 passed; 0 failed; 0 skipped; finished in 52.85ms (52.61ms
CPU time)

Ran 1 test suite in 53.30ms (52.85ms CPU time): 3 tests passed, 0 failed, 0
skipped (3 total tests)
```

Dependencies

Forge manages dependencies using git submodules by default, which means that it works with any GitHub repository that contains smart contracts.

Adding a dependency

To add a dependency, run `forge install`:

```
$ forge install transmissions11/solmate
Installing solmate in /tmp/tmp.kulbTdAAtH/deps/lib/solmate (url:
Some("https://github.com/transmissions11/solmate"), tag: None)
    Installed solmate
```

This pulls the `solmate` library, stages the `.gitmodules` file in git and makes a commit with the message "Installed solmate".

If we now check the `lib` folder:

```
$ tree lib -L 1
lib
├── forge-std
├── solmate
└── weird-erc20

3 directories, 0 files
```

We can see that Forge installed `solmate`!

By default, `forge install` installs the latest master branch version. If you want to install a specific tag or commit, you can do it like so:

```
$ forge install transmissions11/solmate@v7
```

Remapping dependencies

Forge can remap dependencies to make them easier to import. Forge will automatically try to deduce some remappings for you:

```
$ forge remappings
ds-test/=lib/solmate/lib/ds-test/src/
forge-std/=lib/forge-std/src/
solmate/=lib/solmate/src/
weird-erc20/=lib/weird-erc20/src/
```

These remappings mean:

- To import from `forge-std` we would write: `import "forge-std/Contract.sol";`
- To import from `ds-test` we would write: `import "ds-test/Contract.sol";`
- To import from `solmate` we would write: `import "solmate/Contract.sol";`
- To import from `weird-erc20` we would write: `import "weird-erc20/Contract.sol";`

You can customize these remappings by creating a `remappings.txt` file in the root of your project.

Let's create a remapping called `solmate-utils` that points to the `utils` folder in the solmate repository!

```
solmate-utils/=lib/solmate/src/utils/
```

You can also set remappings in `foundry.toml`.

```
remappings = [
    "@solmate-utils/=lib/solmate/src/utils/",
]
```

Now we can import any of the contracts in `src/utils` of the solmate repository like so:

```
import "@solmate-utils/LibString.sol";
```

Updating dependencies

You can update a specific dependency to the latest commit on the version you have specified using `forge update <dep>`. For example, if we wanted to pull the latest commit from our previously installed master-version of `solmate`, we would run:

```
$ forge update lib/solmate
```

Alternatively, you can do this for all dependencies at once by just running `forge update`.

Removing dependencies

You can remove dependencies using `forge remove <deps>...`, where `<deps>` is either the full path to the dependency or just the name. For example, to remove `solmate` both of these commands are equivalent:

```
$ forge remove solmate
# ... is equivalent to ...
$ forge remove lib/solmate
```

Hardhat compatibility

Forge also supports Hardhat-style projects where dependencies are npm packages (stored in `node_modules`) and contracts are stored in `contracts` as opposed to `src`.

To enable Hardhat compatibility mode pass the `--hh` flag.

Project Layout

Forge is flexible on how you structure your project. By default, the structure is:

```
.
├── README.md
├── foundry.toml
├── lib
│   └── forge-std
│       ├── LICENSE-APACHE
│       ├── LICENSE-MIT
│       ├── README.md
│       ├── foundry.toml
│       ├── package.json
│       ├── scripts
│       ├── src
│       └── test
├── script
│   └── Counter.s.sol
├── src
│   └── Counter.sol
└── test
    └── Counter.t.sol

8 directories, 10 files
```

- You can configure Foundry's behavior using `foundry.toml`.
- Remappings are specified in `remappings.txt`.
- The default directory for contracts is `src/`.
- The default directory for tests is `test/`, where any contract with a function that starts with `test` is considered to be a test.
- Dependencies are stored as git submodules in `lib/`.

You can configure where Forge looks for both dependencies and contracts using the `--lib-paths` and `--contracts` flags respectively. Alternatively you can configure it in `foundry.toml`.

Combined with remappings, this gives you the flexibility needed to support the project structure of other toolchains such as Hardhat and Truffle.

For automatic Hardhat support you can also pass the `--hh` flag, which sets the following flags: `--lib-paths node_modules --contracts contracts`.

Overview of Forge

Forge is a command-line tool that ships with Foundry. Forge tests, builds, and deploys your smart contracts.

Tests

Forge can run your tests with the `forge test` command. All tests are written in Solidity.

Forge will look for the tests anywhere in your source directory. Any contract with a function that starts with `test` is considered to be a test. Usually, tests will be placed in `test/` by convention and end with `.t.sol`.

Here's an example of running `forge test` in a freshly created project, that only has the default test:

```
$ forge test
No files changed, compilation skipped

Ran 2 tests for test/Counter.t.sol:CounterTest
[PASS] testFuzz_SetNumber(uint256) (runs: 256, μ: 30688, ~: 31310)
[PASS] test_Increment() (gas: 31325)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 14.43ms (14.04ms
CPU time)

Ran 1 test suite in 15.05ms (14.43ms CPU time): 2 tests passed, 0 failed, 0
skipped (2 total tests)
```

You can also run specific tests by passing a filter:

```
$ forge test --match-contract ComplicatedContractTest --match-test test_Deposit
installing solc version "0.8.10"
Successfully installed solc 0.8.10
Compiling 25 files with 0.8.10
Solc 0.8.10 finished in 1.29s
Compiler run successful!

Ran 2 tests for test/ComplicatedContract.t.sol:ComplicatedContractTest
[PASS] test_DepositERC20() (gas: 102193)
[PASS] test_DepositETH() (gas: 61414)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 1.24ms (1.53ms CPU
time)

Ran 1 test suite in 2.87ms (1.24ms CPU time): 2 tests passed, 0 failed, 0
skipped (2 total tests)
```

This will run the tests in the `ComplicatedContractTest` test contract with `testDeposit` in the name. Inverse versions of these flags also exist (`--no-match-contract` and `--no-match-test`).

You can run tests in filenames that match a glob pattern with `--match-path`.

```
$ forge test --match-path test/ContractB.t.sol
No files changed, compilation skipped

Ran 1 test for test/ContractB.t.sol:ContractBTest
[PASS] testExample() (gas: 257)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 286.23µs (109.21µs
CPU time)

Ran 1 test suite in 2.91ms (286.23µs CPU time): 1 tests passed, 0 failed, 0
skipped (1 total tests)
```

The inverse of the `--match-path` flag is `--no-match-path`.

Logs and traces

The default behavior for `forge test` is to only display a summary of passing and failing tests. You can control this behavior by increasing the verbosity (using the `-v` flag). Each level of verbosity adds more information:

- **Level 2 (`-vv`)**: Logs emitted during tests are also displayed. That includes assertion errors from tests, showing information such as expected vs actual.
- **Level 3 (`-vvv`)**: Stack traces for failing tests are also displayed.
- **Level 4 (`-vvvv`)**: Stack traces for all tests are displayed, and setup traces for failing tests are displayed.
- **Level 5 (`-vvvvv`)**: Stack traces and setup traces are always displayed.

Watch mode

Forge can re-run your tests when you make changes to your files using `forge test --watch`.

By default, only changed test files are re-run. If you want to re-run all tests on a change, you can use `forge test --watch --run-all`.

Writing Tests

Tests are written in Solidity. If the test function reverts, the test fails, otherwise it passes.

Let's go over the most common way of writing tests, using the Forge Standard Library's `Test` contract, which is the preferred way of writing tests with Forge.

In this section, we'll go over the basics using the functions from the Forge Std's `Test` contract, which is itself a superset of DSTest. You will learn how to use more advanced stuff from the Forge Standard Library soon.

DSTest provides basic logging and assertion functionality. To get access to the functions, import `forge-std/Test.sol` and inherit from `Test` in your test contract:

```
import "forge-std/Test.sol";
```

Let's examine a basic test:

```
pragma solidity 0.8.10;

import "forge-std/Test.sol";

contract ContractBTest is Test {
    uint256 testNumber;

    function setUp() public {
        testNumber = 42;
    }

    function test_NumberIs42() public {
        assertEq(testNumber, 42);
    }

    function testFail_Subtract43() public {
        testNumber -= 43;
    }
}
```

Forge uses the following keywords in tests:

- `setUp` : An optional function invoked before each test case is run.

  ```
  function setUp() public {
      testNumber = 42;
  }
  ```

- `test` : Functions prefixed with `test` are run as a test case.

```
function test_NumberIs42() public {
    assertEq(testNumber, 42);
}
```

- `testFail` : The inverse of the `test` prefix - if the function does not revert, the test fails.

```
function testFail_Subtract43() public {
    testNumber -= 43;
}
```

A good practice is to use the pattern `test_Revert[If|When]_Condition` in combination with the `expectRevert` cheatcode (cheatcodes are explained in greater detail in the following section). Also, other testing practices can be found in the Tutorials section. Now, instead of using `testFail` , you know exactly what reverted and with which error:

```
function test_CannotSubtract43() public {
    vm.expectRevert(stdError.arithmeticError);
    testNumber -= 43;
}
```

Tests are deployed to `0xb4c79daB8f259C7Aee6E5b2Aa729821864227e84` . If you deploy a contract within your test, then `0xb4c...7e84` will be its deployer. If the contract deployed within a test gives special permissions to its deployer, such as `Ownable.sol` 's `onlyOwner` modifier, then the test contract `0xb4c...7e84` will have those permissions.

⚠️ **Note**

Test functions must have either `external` or `public` visibility. Functions declared as `internal` or `private` won't be picked up by Forge, even if they are prefixed with `test` .

Shared setups

It is possible to use shared setups by creating helper abstract contracts and inheriting them in your test contracts:

```
abstract contract HelperContract {
    address constant IMPORTANT_ADDRESS = 0x543d...;
    SomeContract someContract;
    constructor() {...}
}

contract MyContractTest is Test, HelperContract {
    function setUp() public {
        someContract = new SomeContract(0, IMPORTANT_ADDRESS);
        ...
    }
}

contract MyOtherContractTest is Test, HelperContract {
    function setUp() public {
        someContract = new SomeContract(1000, IMPORTANT_ADDRESS);
        ...
    }
}
```

💡 **Tip**

Use the `getCode` cheatcode to deploy contracts with incompatible Solidity versions.

Cheatcodes

Most of the time, simply testing your smart contracts outputs isn't enough. To manipulate the state of the blockchain, as well as test for specific reverts and events, Foundry is shipped with a set of cheatcodes.

Cheatcodes allow you to change the block number, your identity, and more. They are invoked by calling specific functions on a specially designated address: `0x7109709ECfa91a80626fF3989D68f67F5b1DD12D`.

You can access cheatcodes easily via the `vm` instance available in Forge Standard Library's `Test` contract. Forge Standard Library is explained in greater detail in the following section.

Let's write a test for a smart contract that is only callable by its owner.

```
pragma solidity 0.8.10;

import "forge-std/Test.sol";

error Unauthorized();

contract OwnerUpOnly {
    address public immutable owner;
    uint256 public count;

    constructor() {
        owner = msg.sender;
    }

    function increment() external {
        if (msg.sender != owner) {
            revert Unauthorized();
        }
        count++;
    }
}

contract OwnerUpOnlyTest is Test {
    OwnerUpOnly upOnly;

    function setUp() public {
        upOnly = new OwnerUpOnly();
    }

    function test_IncrementAsOwner() public {
        assertEq(upOnly.count(), 0);
        upOnly.increment();
        assertEq(upOnly.count(), 1);
    }
}
```

If we run `forge test` now, we will see that the test passes, since `OwnerUpOnlyTest` is the owner of `OwnerUpOnly`.

```
$ forge test
Compiling 26 files with 0.8.10
Solc 0.8.10 finished in 1.51s
Compiler run successful!

Ran 1 test for test/OwnerUpOnly.t.sol:OwnerUpOnlyTest
[PASS] test_IncrementAsOwner() (gas: 29161)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 420.12µs (80.20µs
CPU time)

Ran 1 test suite in 2.79ms (420.12µs CPU time): 1 tests passed, 0 failed, 0
skipped (1 total tests)
```

Let's make sure that someone who is definitely not the owner can't increment the count:

```
contract OwnerUpOnlyTest is Test {
    OwnerUpOnly upOnly;

    // ...

    function testFail_IncrementAsNotOwner() public {
        vm.prank(address(0));
        upOnly.increment();
    }
}
```

If we run `forge test` now, we will see that all the test pass.

```
$ forge test
No files changed, compilation skipped

Ran 2 tests for test/OwnerUpOnly.t.sol:OwnerUpOnlyTest
[PASS] testFail_IncrementAsNotOwner() (gas: 8314)
[PASS] test_IncrementAsOwner() (gas: 29161)
Suite result: ok. 2 passed; 0 failed; 0 skipped; finished in 809.37µs (175.01µs
CPU time)

Ran 1 test suite in 2.92ms (809.37µs CPU time): 2 tests passed, 0 failed, 0
skipped (2 total tests)
```

The test passed because the `prank` cheatcode changed our identity to the zero address for the next call (`upOnly.increment()`). The test case passed since we used the `testFail` prefix, however, using `testFail` is considered an anti-pattern since it does not tell us anything about *why* `upOnly.increment()` reverted.

If we run the tests again with traces turned on, we can see that we reverted with the correct error message.

```
$ forge test -vvvv --match-test testFail_IncrementAsNotOwner
No files changed, compilation skipped

Ran 1 test for test/OwnerUpOnly.t.sol:OwnerUpOnlyTest
[PASS] testFail_IncrementAsNotOwner() (gas: 8314)
Traces:
  [8314] OwnerUpOnlyTest::testFail_IncrementAsNotOwner()
    ├─ [0] VM::prank(0x0000000000000000000000000000000000000000)
    │   └─ ← [Return]
    ├─ [247] OwnerUpOnly::increment()
    │   └─ ← [Revert] Unauthorized()
    └─ ← [Revert] Unauthorized()

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 384.88µs (44.96µs
CPU time)

Ran 1 test suite in 2.83ms (384.88µs CPU time): 1 tests passed, 0 failed, 0
skipped (1 total tests)
```

To be sure in the future, let's make sure that we reverted because we are not the owner using the `expectRevert` cheatcode:

```
contract OwnerUpOnlyTest is Test {
    OwnerUpOnly upOnly;

    // ...

    // Notice that we replaced `testFail` with `test`
    function test_RevertWhen_CallerIsNotOwner() public {
        vm.expectRevert(Unauthorized.selector);
        vm.prank(address(0));
        upOnly.increment();
    }
}
```

If we run `forge test` one last time, we see that the test still passes, but this time we are sure that it will always fail if we revert for any other reason.

```
$ forge test
No files changed, compilation skipped

Ran 1 test for test/OwnerUpOnly.t.sol:OwnerUpOnlyTest
[PASS] test_IncrementAsOwner() (gas: 29161)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 504.30µs (81.72µs
CPU time)

Ran 1 test suite in 2.71ms (504.30µs CPU time): 1 tests passed, 0 failed, 0
skipped (1 total tests)
```

Another cheatcode that is perhaps not so intuitive is the `expectEmit` function. Before looking at `expectEmit`, we need to understand what an event is.

Events are inheritable members of contracts. When you emit an event, the arguments are stored on the blockchain. The `indexed` attribute can be added to a maximum of three parameters of an event to form a data structure known as a "topic." Topics allow users to search for events on the blockchain.

```
pragma solidity 0.8.10;

import "forge-std/Test.sol";

contract EmitContractTest is Test {
    event Transfer(address indexed from, address indexed to, uint256 amount);

    function test_ExpectEmit() public {
        ExpectEmit emitter = new ExpectEmit();
        // Check that topic 1, topic 2, and data are the same as the following
emitted event.
        // Checking topic 3 here doesn't matter, because `Transfer` only has 2
indexed topics.
        vm.expectEmit(true, true, false, true);
        // The event we expect
        emit Transfer(address(this), address(1337), 1337);
        // The event we get
        emitter.t();
    }

    function test_ExpectEmit_DoNotCheckData() public {
        ExpectEmit emitter = new ExpectEmit();
        // Check topic 1 and topic 2, but do not check data
        vm.expectEmit(true, true, false, false);
        // The event we expect
        emit Transfer(address(this), address(1337), 1338);
        // The event we get
        emitter.t();
    }
}

contract ExpectEmit {
    event Transfer(address indexed from, address indexed to, uint256 amount);

    function t() public {
        emit Transfer(msg.sender, address(1337), 1337);
    }
}
```

When we call `vm.expectEmit(true, true, false, true);`, we want to check the 1st and 2nd `indexed` topic for the next event.

The expected `Transfer` event in `test_ExpectEmit()` means we are expecting that `from` is `address(this)`, and `to` is `address(1337)`. This is compared against the event emitted from `emitter.t()`.

In other words, we are checking that the first topic from `emitter.t()` is equal to `address(this)`. The 3rd argument in `expectEmit` is set to `false` because there is no need

to check the third topic in the `Transfer` event, since there are only two. It does not matter even if we set to `true`.

The 4th argument in `expectEmit` is set to `true`, which means that we want to check "non-indexed topics", also known as data.

For example, we want the data from the expected event in `test_ExpectEmit` - which is `amount` - to equal to the data in the actual emitted event. In other words, we are asserting that `amount` emitted by `emitter.t()` is equal to `1337`. If the fourth argument in `expectEmit` was set to `false`, we would not check `amount`.

In other words, `test_ExpectEmit_DoNotCheckData` is a valid test case, even though the amounts differ, since we do not check the data.

📚 Reference

See the Cheatcodes Reference for a complete overview of all the available cheatcodes.

Forge Standard Library Overview

Forge Standard Library (Forge Std for short) is a collection of helpful contracts that make writing tests easier, faster, and more user-friendly.

Using Forge Std is the preferred way of writing tests with Foundry.

It provides all the essential functionality you need to get started writing tests:

- `Vm.sol` : Up-to-date cheatcodes interface
- `console.sol` and `console2.sol` : Hardhat-style logging functionality
- `Script.sol` : Basic utilities for Solidity scripting
- `Test.sol` : A superset of DSTest containing standard libraries, a cheatcodes instance (`vm`), and Hardhat console

Simply import `Test.sol` and inherit from `Test` in your test contract:

```
import "forge-std/Test.sol";

contract ContractTest is Test { ...
```

Now, you can:

```
// Access Hevm via the `vm` instance
vm.startPrank(alice);

// Assert and log using Dappsys Test
assertEq(dai.balanceOf(alice), 10000e18);

// Log with the Hardhat `console` (`console2`)
console.log(alice.balance);

// Use anything from the Forge Std std-libraries
deal(address(dai), alice, 10000e18);
```

To import the `Vm` interface or the `console` library individually:

```
import "forge-std/Vm.sol";


import "forge-std/console.sol";
```

Note: `console2.sol` contains patches to `console.sol` that allows Forge to decode traces for calls to the console, but it is not compatible with Hardhat.

```
import "forge-std/console2.sol";
```

Standard libraries

Forge Std currently consists of six standard libraries.

Std Logs

Std Logs expand upon the logging events from the `DSTest` library.

Std Assertions

Std Assertions expand upon the assertion functions from the `DSTest` library.

Std Cheats

Std Cheats are wrappers around Forge cheatcodes that make them safer to use and improve the DX.

You can access Std Cheats by simply calling them inside your test contract, as you would any other internal function:

```
// set up a prank as Alice with 100 ETH balance
hoax(alice, 100 ether);
```

Std Errors

Std Errors provide wrappers around common internal Solidity errors and reverts.

Std Errors are most useful in combination with the `expectRevert` cheatcode, as you do not need to remember the internal Solidity panic codes yourself. Note that you have to access them through `stdError`, as this is a library.

```
// expect an arithmetic error on the next call (e.g. underflow)
vm.expectRevert(stdError.arithmeticError);
```

Std Storage

Std Storage makes manipulating contract storage easy. It can find and write to the storage slot(s) associated with a particular variable.

The `Test` contract already provides a `StdStorage` instance `stdstore` through which you can access any std-storage functionality. Note that you must add `using stdStorage for StdStorage` in your test contract first.

```
// find the variable `score` in the contract `game`
// and change its value to 10
stdstore
    .target(address(game))
    .sig(game.score.selector)
    .checked_write(10);
```

Std Math

Std Math is a library with useful mathematical functions that are not provided in Solidity.

Note that you have to access them through `stdMath`, as this is a library.

```
// get the absolute value of -10
uint256 ten = stdMath.abs(-10)
```

📚 **Reference**

See the Forge Standard Library Reference for a complete overview of Forge Standard Library.

Understanding Traces

Forge can produce traces either for failing tests (`-vvv`) or all tests (`-vvvv`).

Traces follow the same general format:

```
[<Gas Usage>] <Contract>::<Function>(<Parameters>)
  ├─ [<Gas Usage>] <Contract>::<Function>(<Parameters>)
  │   └─ ← <Return Value>
  └─ ← <Return Value>
```

Each trace can have many more subtraces, each denoting a call to a contract and a return value.

If your terminal supports color, the traces will also come with a variety of colors:

- **Green**: For calls that do not revert
- **Red**: For reverting calls
- **Blue**: For calls to cheat codes
- **Cyan**: For emitted logs
- **Yellow**: For contract deployments

The gas usage (marked in square brackets) is for the entirety of the function call. You may notice, however, that sometimes the gas usage of one trace does not exactly match the gas usage of all its subtraces:

```
[24661] OwnerUpOnlyTest::testIncrementAsOwner()
  ├─ [2262] OwnerUpOnly::count()
  │   └─ ← 0
  ├─ [20398] OwnerUpOnly::increment()
  │   └─ ← ()
  ├─ [262] OwnerUpOnly::count()
  │   └─ ← 1
  └─ ← ()
```

The gas unaccounted for is due to some extra operations happening between calls, such as arithmetic and store reads/writes.

Forge will try to decode as many signatures and values as possible, but sometimes this is not possible. In these cases, the traces will appear like so:

```
[<Gas Usage>] <Address>::<Calldata>
  └─ ← <Return Data>
```

Fork Testing

Forge supports testing in a forked environment with two different approaches:

- **Forking Mode** — use a single fork for all your tests via the `forge test --fork-url` flag
- **Forking Cheatcodes** — create, select, and manage multiple forks directly in Solidity test code via forking cheatcodes

Which approach to use? Forking mode affords running an entire test suite against a specific forked environment, while forking cheatcodes provide more flexibility and expressiveness to work with multiple forks in your tests. Your particular use case and testing strategy will help inform which approach to use.

Forking Mode

To run all tests in a forked environment, such as a forked Ethereum mainnet, pass an RPC URL via the `--fork-url` flag:

```
forge test --fork-url <your_rpc_url>
```

The following values are changed to reflect those of the chain at the moment of forking:

- `block_number`
- `chain_id`
- `gas_limit`
- `gas_price`
- `block_base_fee_per_gas`
- `block_coinbase`
- `block_timestamp`
- `block_difficulty`

It is possible to specify a block from which to fork with `--fork-block-number`:

```
forge test --fork-url <your_rpc_url> --fork-block-number 1
```

Forking is especially useful when you need to interact with existing contracts. You may choose to do integration testing this way, as if you were on an actual network.

Caching

If both `--fork-url` and `--fork-block-number` are specified, then data for that block is cached for future test runs.

The data is cached in `~/.foundry/cache/rpc/<chain name>/<block number>`. To clear the cache, simply remove the directory or run `forge clean` (removes all build artifacts and cache directories).

It is also possible to ignore the cache entirely by passing `--no-storage-caching`, or with `foundry.toml` by configuring `no_storage_caching` and `rpc_storage_caching`.

Improved traces

Forge supports identifying contracts in a forked environment with Etherscan.

To use this feature, pass the Etherscan API key via the `--etherscan-api-key` flag:

```
forge test --fork-url <your_rpc_url> --etherscan-api-key
<your_etherscan_api_key>
```

Alternatively, you can set the `ETHERSCAN_API_KEY` environment variable.

Forking Cheatcodes

Forking cheatcodes allow you to enter forking mode programmatically in your Solidity test code. Instead of configuring forking mode via `forge` CLI arguments, these cheatcodes allow you to use forking mode on a test-by-test basis and work with multiple forks in your tests. Each fork is identified via its own unique `uint256` identifier.

Usage

Important to keep in mind that *all* test functions are isolated, meaning each test function is executed with a *copy* of the state *after* `setUp` and is executed in its own stand-alone EVM.

Therefore forks created during `setUp` are available in tests. The code example below uses `createFork` to create two forks, but does *not* select one initially. Each fork is identified with a unique identifier (`uint256 forkId`), which is assigned when it is first created.

Enabling a specific fork is done via passing that `forkId` to `selectFork`.

`createSelectFork` is a one-liner for `createFork` plus `selectFork`.

There can only be one fork active at a time, and the identifier for the currently active fork can be retrieved via `activeFork`.

Similar to `roll`, you can set `block.number` of a fork with `rollFork`.

To understand what happens when a fork is selected, it is important to know how the forking mode works in general:

Each fork is a standalone EVM, i.e. all forks use completely independent storage. The only exception is the state of the `msg.sender` and the test contract itself, which are persistent across fork swaps. In other words all changes that are made while fork `A` is active (`selectFork(A)`) are only recorded in fork `A`'s storage and are not available if another fork is selected. However, changes recorded in the test contract itself (variables) are still available because the test contract is a *persistent* account.

The `selectFork` cheatcode sets the *remote* section with the fork's data source, however the *local* memory remains persistent across fork swaps. This also means `selectFork` can be called at all times with any fork, to set the *remote* data source. However, it is important to keep in mind the above rules for `read/write` access always apply, meaning *writes* are persistent across fork swaps.

Examples

Create and Select Forks

```
contract ForkTest is Test {
    // the identifiers of the forks
    uint256 mainnetFork;
    uint256 optimismFork;

    //Access variables from .env file via vm.envString("varname")
    //Replace ALCHEMY_KEY by your alchemy key or Etherscan key, change RPC url
if need
    //inside your .env file e.g:
    //MAINNET_RPC_URL = 'https://eth-mainnet.g.alchemy.com/v2/ALCHEMY_KEY'
    //string MAINNET_RPC_URL = vm.envString("MAINNET_RPC_URL");
    //string OPTIMISM_RPC_URL = vm.envString("OPTIMISM_RPC_URL");

    // create two _different_ forks during setup
    function setUp() public {
        mainnetFork = vm.createFork(MAINNET_RPC_URL);
        optimismFork = vm.createFork(OPTIMISM_RPC_URL);
    }

    // demonstrate fork ids are unique
    function testForkIdDiffer() public {
        assert(mainnetFork != optimismFork);
    }

    // select a specific fork
    function testCanSelectFork() public {
        // select the fork
        vm.selectFork(mainnetFork);
        assertEq(vm.activeFork(), mainnetFork);

        // from here on data is fetched from the `mainnetFork` if the EVM
requests it and written to the storage of `mainnetFork`
    }

    // manage multiple forks in the same test
    function testCanSwitchForks() public {
        vm.selectFork(mainnetFork);
        assertEq(vm.activeFork(), mainnetFork);

        vm.selectFork(optimismFork);
        assertEq(vm.activeFork(), optimismFork);
    }

    // forks can be created at all times
    function testCanCreateAndSelectForkInOneStep() public {
        // creates a new fork and also selects it
        uint256 anotherFork = vm.createSelectFork(MAINNET_RPC_URL);
        assertEq(vm.activeFork(), anotherFork);
    }

    // set `block.number` of a fork
    function testCanSetForkBlockNumber() public {
```

```
        vm.selectFork(mainnetFork);
        vm.rollFork(1_337_000);

        assertEq(block.number, 1_337_000);
    }
}
```

Separated and persistent storage

As mentioned each fork is essentially an independent EVM with separated storage.

Only the accounts of `msg.sender` and the test contract (`ForkTest`) are persistent when forks are selected. But any account can be turned into a persistent account: `makePersistent` .

An account that is *persistent* is unique, i.e. it exists on all forks

```
contract ForkTest is Test {
    // the identifiers of the forks
    uint256 mainnetFork;
    uint256 optimismFork;

    //Access variables from .env file via vm.envString("varname")
    //Replace ALCHEMY_KEY by your alchemy key or Etherscan key, change RPC url
if need
    //inside your .env file e.g:
    //MAINNET_RPC_URL = 'https://eth-mainnet.g.alchemy.com/v2/ALCHEMY_KEY'
    //string MAINNET_RPC_URL = vm.envString("MAINNET_RPC_URL");
    //string OPTIMISM_RPC_URL = vm.envString("OPTIMISM_RPC_URL");

    // create two _different_ forks during setup
    function setUp() public {
        mainnetFork = vm.createFork(MAINNET_RPC_URL);
        optimismFork = vm.createFork(OPTIMISM_RPC_URL);
    }

    // creates a new contract while a fork is active
    function testCreateContract() public {
        vm.selectFork(mainnetFork);
        assertEq(vm.activeFork(), mainnetFork);

        // the new contract is written to `mainnetFork`'s storage
        SimpleStorageContract simple = new SimpleStorageContract();

        // and can be used as normal
        simple.set(100);
        assertEq(simple.value(), 100);

        // after switching to another contract we still know `address(simple)`
but the contract only lives in `mainnetFork`
        vm.selectFork(optimismFork);

        /* this call will therefore revert because `simple` now points to a
contract that does not exist on the active fork
        * it will produce following revert message:
        *
        * "Contract 0xCe71065D4017F316EC606Fe4422e11eB2c47c246 does not exist
on active fork with id `1`
        *       But exists on non active forks: `[0]`"
        */
        simple.value();
    }

     // creates a new _persistent_ contract while a fork is active
     function testCreatePersistentContract() public {
        vm.selectFork(mainnetFork);
        SimpleStorageContract simple = new SimpleStorageContract();
        simple.set(100);
        assertEq(simple.value(), 100);

        // mark the contract as persistent so it is also available when other
forks are active
        vm.makePersistent(address(simple));
        assert(vm.isPersistent(address(simple)));
```

```
        vm.selectFork(optimismFork);
        assert(vm.isPersistent(address(simple)));

        // This will succeed because the contract is now also available on the
`optimismFork`
        assertEq(simple.value(), 100);
    }
}

contract SimpleStorageContract {
    uint256 public value;

    function set(uint256 _value) public {
        value = _value;
    }
}
```

For more details and examples, see the forking cheatcodes reference.

Advanced Testing

Forge comes with a number of advanced testing methods:

- Fuzz Testing
- Invariant Testing
- Differential Testing

In the future, Forge will also support these:

- Symbolic Execution
- Mutation Testing

Each chapter dives into what problem the testing methods solve, and how to apply them to your own project.

Fuzz Testing

Forge supports property based testing.

Property-based testing is a way of testing general behaviors as opposed to isolated scenarios.

Let's examine what that means by writing a unit test, finding the general property we are testing for, and converting it to a property-based test instead:

```
pragma solidity 0.8.10;

import "forge-std/Test.sol";

contract Safe {
    receive() external payable {}

    function withdraw() external {
        payable(msg.sender).transfer(address(this).balance);
    }
}

contract SafeTest is Test {
    Safe safe;

    // Needed so the test contract itself can receive ether
    // when withdrawing
    receive() external payable {}

    function setUp() public {
        safe = new Safe();
    }

    function test_Withdraw() public {
        payable(address(safe)).transfer(1 ether);
        uint256 preBalance = address(this).balance;
        safe.withdraw();
        uint256 postBalance = address(this).balance;
        assertEq(preBalance + 1 ether, postBalance);
    }
}
```

Running the test, we see it passes:

```
$ forge test
Compiling 24 files with 0.8.10
Solc 0.8.10 finished in 1.17s
Compiler run successful!

Ran 1 test for test/Safe.t.sol:SafeTest
[PASS] test_Withdraw() (gas: 19463)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 520.18µs (110.20µs
CPU time)

Ran 1 test suite in 2.82ms (520.18µs CPU time): 1 tests passed, 0 failed, 0
skipped (1 total tests)
```

This unit test *does test* that we can withdraw ether from our safe. However, who is to say that it works for all amounts, not just 1 ether?

The general property here is: given a safe balance, when we withdraw, we should get whatever is in the safe.

Forge will run any test that takes at least one parameter as a property-based test, so let's rewrite:

```
contract SafeTest is Test {
    // ...

    function testFuzz_Withdraw(uint256 amount) public {
        payable(address(safe)).transfer(amount);
        uint256 preBalance = address(this).balance;
        safe.withdraw();
        uint256 postBalance = address(this).balance;
        assertEq(preBalance + amount, postBalance);
    }
}
```

If we run the test now, we can see that Forge runs the property-based test, but it fails for high values of `amount`:

```
$ forge test
Compiling 1 files with 0.8.10
Solc 0.8.10 finished in 1.10s
Compiler run successful!

Ran 1 test for test/Safe.t.sol:SafeTest
[FAIL. Reason: EvmError: Revert; counterexample:
calldata=0x29facca70000000000000000000000000000000000000100000000000000000000
0000 args=[7922816251426433759354395033 6 [7.922e28]]]
testFuzz_Withdraw(uint256) (runs: 3, µ: 19531, ~: 19531)
Suite result: FAILED. 0 passed; 1 failed; 0 skipped; finished in 5.67ms (5.30ms
CPU time)

Ran 1 test suite in 6.44ms (5.67ms CPU time): 0 tests passed, 1 failed, 0
skipped (1 total tests)
```

The default amount of ether that the test contract is given is `2**96 wei` (as in DappTools), so we have to restrict the type of amount to `uint96` to make sure we don't try to send more than we have:

```
function testFuzz_Withdraw(uint96 amount) public {
```

And now it passes:

```
$ forge test
Compiling 1 files with 0.8.10
Solc 0.8.10 finished in 1.10s
Compiler run successful!

Ran 1 test for test/Safe.t.sol:SafeTest
[PASS] testFuzz_Withdraw(uint96) (runs: 257, μ: 19109, ~: 19631)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 15.31ms (14.84ms
CPU time)

Ran 1 test suite in 16.05ms (15.31ms CPU time): 1 tests passed, 0 failed, 0
skipped (1 total tests)
```

You may want to exclude certain cases using the `assume` cheatcode. In those cases, fuzzer will discard the inputs and start a new fuzz run:

```
function testFuzz_Withdraw(uint96 amount) public {
    vm.assume(amount > 0.1 ether);
    // snip
}
```

There are different ways to run property-based tests, notably parametric testing and fuzzing. Forge only supports fuzzing.

Interpreting results

You might have noticed that fuzz tests are summarized a bit differently compared to unit tests:

- "runs" refers to the amount of scenarios the fuzzer tested. By default, the fuzzer will generate 256 scenarios, but this and other test execution parameters can be setup by the user. Fuzzer configuration details are provided `here`.
- "μ" (Greek letter mu) is the mean gas used across all fuzz runs
- "~" (tilde) is the median gas used across all fuzz runs

Configuring fuzz test execution

Fuzz tests execution is governed by parameters that can be controlled by users via Forge configuration primitives. Configs can be applied globally or on a per-test basis. For details on this topic please refer to `Global config` and `In-line config`.

Invariant Testing

Overview

Invariant testing allows for a set of invariant expressions to be tested against randomized sequences of pre-defined function calls from pre-defined contracts. After each function call is performed, all defined invariants are asserted.

Invariant testing is a powerful tool to expose incorrect logic in protocols. Due to the fact that function call sequences are randomized and have fuzzed inputs, invariant testing can expose false assumptions and incorrect logic in edge cases and highly complex protocol states.

Invariant testing campaigns have two dimensions, `runs` and `depth`:

- `runs`: Number of times that a sequence of function calls is generated and run.
- `depth`: Number of function calls made in a given `run`. All defined invariants are asserted after each function call is made. If a function call reverts, the `depth` counter still increments.

These and other invariant configuration aspects are explained `here`.

Similar to how standard tests are run in Foundry by prefixing a function name with `test`, invariant tests are denoted by prefixing the function name with `invariant` (e.g., `function invariant_A()`).

Configuring invariant test execution

Invariant tests execution is governed by parameters that can be controlled by users via Forge configuration primitives. Configs can be applied globally or on a per-test basis. For details on this topic please refer to 📚 `Global config` and 📚 `In-line config`.

Defining Invariants

Invariants are conditions expressions that should always hold true over the course of a fuzzing campaign. A good invariant testing suite should have as many invariants as possible, and can have different testing suites for different protocol states.

Examples of invariants are:

- *"The xy=k formula always holds"* for Uniswap
- *"The sum of all user balances is equal to the total supply"* for an ERC-20 token.

There are different ways to assert invariants, as outlined in the table below:

Type	Explanation	Example

Direct assertions	Query a protocol smart contract and assert values are as expected.	`assertGe(`
		`token.totalAssets(),`
		`token.totalSupply()`
		`)`

Ghost variable assertions	Query a protocol smart contract and compare it against a value that has been persisted in the test environment (ghost variable).	`assertEq(`
		`token.totalSupply(),`
		`sumBalanceOf`
		`)`

Deoptimizing (Naive implementation assertions)	Query a protocol smart contract and compare it against a naive and typically highly gas-inefficient implementation of the same desired logic.	`assertEq(`
		`pool.outstandingInterest(),`
		`test.naiveInterest()`
		`)`

Conditional Invariants

Invariants must hold over the course of a given fuzzing campaign, but that doesn't mean they must hold true in every situation. There is the possibility for certain invariants to be introduced/removed in a given scenario (e.g., during a liquidation).

It is not recommended to introduce conditional logic into invariant assertions because they have the possibility of introducing false positives because of an incorrect code path. For example:

```
function invariant_example() external {
    if (protocolCondition) return;

    assertEq(val1, val2);
}
```

In this situation, if `protocolCondition == true`, the invariant is not asserted at all. Sometimes this can be desired behavior, but it can cause issues if the `protocolCondition` is true for the whole fuzzing campaign unexpectedly, or there is a logic error in the condition

itself. For this reason its better to try and define an alternative invariant for that condition as well, for example:

```
function invariant_example() external {
    if (protocolCondition) {
        assertLe(val1, val2);
        return;
    };

    assertEq(val1, val2);
}
```

Another approach to handle different invariants across protocol states is to utilize dedicated invariant testing contracts for different scenarios. These scenarios can be bootstrapped using the `setUp` function, but it is more powerful to leverage *invariant targets* to govern the fuzzer to behave in a way that will only yield certain results (e.g., avoid liquidations).

Invariant Targets

Target Contracts: The set of contracts that will be called over the course of a given invariant test fuzzing campaign. This set of contracts defaults to all contracts that were deployed in the `setUp` function, but can be customized to allow for more advanced invariant testing.

Target Senders: The invariant test fuzzer picks values for `msg.sender` at random when performing fuzz campaigns to simulate multiple actors in a system by default. If desired, the set of senders can be customized in the `setUp` function.

Target Selectors: The set of function selectors that are used by the fuzzer for invariant testing. These can be used to use a subset of functions within a given target contract.

Target Artifacts: The desired ABI to be used for a given contract. These can be used for proxy contract configurations.

Target Artifact Selectors: The desired subset of function selectors to be used within a given ABI to be used for a given contract. These can be used for proxy contract configurations.

Priorities for the invariant fuzzer in the cases of target clashes are:

`targetSelectors | targetArtifactSelectors > excludeContracts | excludeArtifacts > targetContracts | targetArtifacts`

Function Call Probability Distribution

Functions from these contracts will be called at random with fuzzed inputs. The probability of a function being called is broken down by contract and then by function.

For example:

```
targetContract1: 50%
├─ function1: 50% (25%)
└─ function2: 50% (25%)

targetContract2: 50%
├─ function1: 25% (12.5%)
├─ function2: 25% (12.5%)
├─ function3: 25% (12.5%)
└─ function4: 25% (12.5%)
```

This is something to be mindful of when designing target contracts, as target contracts with less functions will have each function called more often due to this probability distribution.

Invariant Test Helper Functions

Invariant test helper functions are included in `forge-std` to allow for configurable invariant test setup. The helper functions are outlined below:

Function	Description
`excludeContract(address newExcludedContract_)`	Adds a given address to the `_excludedContracts` array. This set of contracts is explicitly excluded from the target contracts.
`excludeSender(address newExcludedSender_)`	Adds a given address to the `_excludedSenders` array. This set of addresses is explicitly excluded from the target senders.
`excludeArtifact(string memory newExcludedArtifact_)`	Adds a given string to the `_excludedArtifacts` array. This set of strings is explicitly excluded from the target artifacts.
`targetArtifact(string memory newTargetedArtifact_)`	Adds a given string to the `_targetedArtifacts` array. This set of strings is used for the target artifacts.
`targetArtifactSelector(FuzzSelector memory newTargetedArtifactSelector_)`	Adds a given `FuzzSelector` to the to `_targetedArtifactSelectors` array. This set of `FuzzSelector`s is used for the target artifact selectors.

Function	Description
`targetContract(address newTargetedContract_)`	Adds a given address to the to `_targetedContracts` array. This set of addresses is used for the target contracts. This array overwrites the set of contracts that was deployed during the `setUp`.
`targetSelector(FuzzSelector memory newTargetedSelector_)`	Adds a given `FuzzSelector` to the to `_targetedSelectors` array. This set of `FuzzSelector`s is used for the target contract selectors.
`targetSender(address newTargetedSender_)`	Adds a given address to the to `_targetedSenders` array. This set of addresses is used for the target senders.

Target Contract Setup

Target contracts can be set up using the following three methods:

1. Contracts that are manually added to the `targetContracts` array are added to the set of target contracts.
2. Contracts that are deployed in the `setUp` function are automatically added to the set of target contracts (only works if no contracts have been manually added using option 1).
3. Contracts that are deployed in the `setUp` can be **removed** from the target contracts if they are added to the `excludeContracts` array.

Open Testing

The default configuration for target contracts is set to all contracts that are deployed during the setup. For smaller modules and more arithmetic contracts, this works well. For example:

```
contract ExampleContract1 {

    uint256 public val1;
    uint256 public val2;
    uint256 public val3;

    function addToA(uint256 amount) external {
        val1 += amount;
        val3 += amount;
    }

    function addToB(uint256 amount) external {
        val2 += amount;
        val3 += amount;
    }

}
```

This contract could be deployed and tested using the default target contract pattern:

```
contract InvariantExample1 is Test {

    ExampleContract1 foo;

    function setUp() external {
        foo = new ExampleContract1();
    }

    function invariant_A() external {
        assertEq(foo.val1() + foo.val2(), foo.val3());
    }

    function invariant_B() external {
        assertGe(foo.val1() + foo.val2(), foo.val3());
    }

}
```

This setup will call `foo.addToA()` and `foo.addToB()` with a 50%-50% probability distribution with fuzzed inputs. Inevitably, the inputs will start to cause overflows and the function calls will start reverting. Since the default configuration in invariant testing is `fail_on_revert = false`, this will not cause the tests to fail. The invariants will hold throughout the rest of the fuzzing campaign and the result is that the test will pass. The output will look something like this:

```
[PASS] invariant_A() (runs: 50, calls: 10000, reverts: 5533)
[PASS] invariant_B() (runs: 50, calls: 10000, reverts: 5533)
```

Handler-Based Testing

For more complex and integrated protocols, more sophisticated target contract usage is required to achieve the desired results. To illustrate how Handlers can be leveraged, the following contract will be used (an ERC-4626 based contract that accepts deposits of another ERC-20 token):

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.17;

interface IERC20Like {

    function balanceOf(address owner_) external view returns (uint256
balance_);

    function transferFrom(
        address owner_,
        address recipient_,
        uint256 amount_
    ) external returns (bool success_);

}

contract Basic4626Deposit {


/*****************************************************************************
****************/
    /*** Storage
***/

/*****************************************************************************
****************/

    address public immutable asset;

    string public name;
    string public symbol;

    uint8 public immutable decimals;

    uint256 public totalSupply;

    mapping(address => uint256) public balanceOf;


/*****************************************************************************
****************/
    /*** Constructor
***/

/*****************************************************************************
****************/

    constructor(address asset_, string memory name_, string memory symbol_,
uint8 decimals_) {
        asset    = asset_;
        name     = name_;
        symbol   = symbol_;
        decimals = decimals_;
    }


/*****************************************************************************
```

```
****************/
    /*** External Functions
***/

/*****************************************************************************
****************/

    function deposit(uint256 assets_, address receiver_) external returns
(uint256 shares_) {
        shares_ = convertToShares(assets_);

        require(receiver_ != address(0), "ZERO_RECEIVER");
        require(shares_   != uint256(0), "ZERO_SHARES");
        require(assets_   != uint256(0), "ZERO_ASSETS");

        totalSupply += shares_;

        // Cannot overflow because totalSupply would first overflow in the
statement above.
        unchecked { balanceOf[receiver_] += shares_; }

        require(
            IERC20Like(asset).transferFrom(msg.sender, address(this), assets_),
            "TRANSFER_FROM"
        );
    }

    function transfer(address recipient_, uint256 amount_) external returns
(bool success_) {
        balanceOf[msg.sender] -= amount_;

        // Cannot overflow because minting prevents overflow of totalSupply,
        // and sum of user balances == totalSupply.
        unchecked { balanceOf[recipient_] += amount_; }

        return true;
    }


/*****************************************************************************
****************/
    /*** Public View Functions
***/

/*****************************************************************************
****************/

    function convertToShares(uint256 assets_) public view returns (uint256
shares_) {
        uint256 supply_ = totalSupply;  // Cache to stack.

        shares_ = supply_ == 0 ? assets_ : (assets_ * supply_) / totalAssets();
    }

    function totalAssets() public view returns (uint256 assets_) {
        assets_ = IERC20Like(asset).balanceOf(address(this));
    }
```

```
}
```

Handler Functions

This contract's `deposit` function requires that the caller has a non-zero balance of the ERC-20 `asset`. In the Open invariant testing approach, `deposit()` and `transfer()` would be called with a 50-50% distribution, but they would revert on every call. This would cause the invariant tests to "pass", but in reality no state was manipulated in the desired contract at all. This is where target contracts can be leveraged. When a contract requires some additional logic in order to function properly, it can be added in a dedicated contract called a `Handler`.

```
function deposit(uint256 assets) public virtual {
    asset.mint(address(this), assets);

    asset.approve(address(token), assets);

    uint256 shares = token.deposit(assets, address(this));
}
```

This contract will provide the necessary setup before a function call is made in order to ensure it is successful.

Building on this concept, Handlers can be used to develop more sophisticated invariant tests. With Open invariant testing, the tests run as shown in the diagram below, with random sequences of function calls being made to the protocol contracts directly with fuzzed parameters. This will cause reverts for more complex systems as outlined above.

By manually adding all Handler contracts to the `targetContracts` array, all function calls made to protocol contracts can be made in a way that is governed by the Handler to ensure successful calls. This is outlined in the diagram below.

With this layer between the fuzzer and the protocol, more powerful testing can be achieved.

Handler Ghost Variables

Within Handlers, "ghost variables" can be tracked across multiple function calls to add additional information for invariant tests. A good example of this is summing all of the `shares` that each LP owns after depositing into the ERC-4626 token as shown above, and using that in the invariant (`totalSupply == sumBalanceOf`).

```
function deposit(uint256 assets) public virtual {
    asset.mint(address(this), assets);

    asset.approve(address(token), assets);

    uint256 shares = token.deposit(assets, address(this));

    sumBalanceOf += shares;
}
```

Function-Level Assertions

Another benefit is the ability to perform assertions on function calls as they are happening. An example is asserting the ERC-20 balance of the LP has decremented by `assets` during the `deposit` function call, as well as their LP token balance incrementing by `shares`. In this way, handler functions are similar to fuzz tests because they can take in fuzzed inputs, perform state changes, and assert before/after state.

```
function deposit(uint256 assets) public virtual {
    asset.mint(address(this), assets);

    asset.approve(address(token), assets);

    uint256 beforeBalance = asset.balanceOf(address(this));

    uint256 shares = token.deposit(assets, address(this));

    assertEq(asset.balanceOf(address(this)), beforeBalance - assets);

    sumBalanceOf += shares;
}
```

Bounded/Unbounded Functions

In addition, with Handlers, input parameters can be bounded to reasonable expected values such that `fail_on_revert` in `foundry.toml` can be set to `true`. This can be accomplished using the `bound()` helper function from `forge-std`. This ensures that every function call that is being made by the fuzzer must be successful against the protocol in order to get tests to pass. This is very useful for visibility and confidence that the protocol is being tested in the desired way.

```
function deposit(uint256 assets) external {
    assets = bound(assets, 0, 1e30);

    asset.mint(address(this), assets);

    asset.approve(address(token), assets);

    uint256 beforeBalance = asset.balanceOf(address(this));

    uint256 shares = token.deposit(assets, address(this));

    assertEq(asset.balanceOf(address(this)), beforeBalance - assets);

    sumBalanceOf += shares;
}
```

This can also be accomplished by inheriting non-bounded functions from dedicated "unbounded" Handler contracts that can be used for `fail_on_revert = false` testing. This

type of testing is also useful since it can expose issues in assumptions made with `bound` function usage.

```
// Unbounded
function deposit(uint256 assets) public virtual {
    asset.mint(address(this), assets);

    asset.approve(address(token), assets);

    uint256 beforeBalance = asset.balanceOf(address(this));

    uint256 shares = token.deposit(assets, address(this));

    assertEq(asset.balanceOf(address(this)), beforeBalance - assets);

    sumBalanceOf += shares;
}


// Bounded
function deposit(uint256 assets) external {
    assets = bound(assets, 0, 1e30);

    super.deposit(assets);
}
```

Actor Management

In the function calls above, it can be seen that `address(this)` is the sole depositor in the ERC-4626 contract, which is not a realistic representation of its intended use. By leveraging the `prank` cheatcodes in `forge-std`, each Handler can manage a set of actors and use them to perform the same function call from different `msg.sender` addresses. This can be accomplished using the following modifier:

```
address[] public actors;

address internal currentActor;

modifier useActor(uint256 actorIndexSeed) {
    currentActor = actors[bound(actorIndexSeed, 0, actors.length - 1)];
    vm.startPrank(currentActor);
    _;
    vm.stopPrank();
}
```

Using multiple actors allows for more granular ghost variable usage as well. This is demonstrated in the functions below:

```
// Unbounded
function deposit(
    uint256 assets,
    uint256 actorIndexSeed
) public virtual useActor(actorIndexSeed) {
    asset.mint(currentActor, assets);

    asset.approve(address(token), assets);

    uint256 beforeBalance = asset.balanceOf(address(this));

    uint256 shares = token.deposit(assets, address(this));

    assertEq(asset.balanceOf(address(this)), beforeBalance - assets);

    sumBalanceOf += shares;

    sumDeposits[currentActor] += assets
}


// Bounded
function deposit(uint256 assets, uint256 actorIndexSeed) external {
    assets = bound(assets, 0, 1e30);

    super.deposit(assets, actorIndexSeed);
}
```

Differential Testing

Forge can be used for differential testing and differential fuzzing. You can even test against non-EVM executables using the `ffi` cheatcode.

Background

Differential testing cross references multiple implementations of the same function by comparing each one's output. Imagine we have a function specification `F(X)`, and two implementations of that specification: `f1(X)` and `f2(X)`. We expect `f1(x) == f2(x)` for all x that exist in an appropriate input space. If `f1(x) != f2(x)`, we know that at least one function is incorrectly implementing `F(X)`. This process of testing for equality and identifying discrepancies is the core of differential testing.

Differential fuzzing is an extension of differential testing. Differential fuzzing programmatically generates many values of `x` to find discrepancies and edge cases that manually chosen inputs might not reveal.

> Note: the `==` operator in this case can be a custom definition of equality. For example, if testing floating point implementations, you might use approximate equality with a certain tolerance.

Some real life uses of this type of testing include:

- Comparing upgraded implementations to their predecessors
- Testing code against known reference implementations
- Confirming compatibility with third party tools and dependencies

Below are some examples of how Forge is used for differential testing.

Primer: The `ffi` cheatcode

`ffi` allows you to execute an arbitrary shell command and capture the output. Here's a mock example:

```
import "forge-std/Test.sol";

contract TestContract is Test {

    function testMyFFI () public {
        string[] memory cmds = new string[](2);
        cmds[0] = "cat";
        cmds[1] = "address.txt"; // assume contains abi-encoded address.
        bytes memory result = vm.ffi(cmds);
        address loadedAddress = abi.decode(result, (address));
        // Do something with the address
        // ...
    }
}
```

An address has previously been written to `address.txt`, and we read it in using the FFI cheatcode. This data can now be used throughout your test contract.

Example: Differential Testing Merkle Tree Implementations

Merkle Trees are a cryptographic commitment scheme frequently used in blockchain applications. Their popularity has led to a number of different implementations of Merkle Tree generators, provers, and verifiers. Merkle roots and proofs are often generated using a language like JavaScript or Python, while proof verification usually occurs on-chain in Solidity.

Murky is a complete implementation of Merkle roots, proofs, and verification in Solidity. Its test suite includes differential tests against OpenZeppelin's Merkle proof library, as well as root generation tests against a reference JavaScript implementation. These tests are powered by Foundry's fuzzing and `ffi` capabilities.

Differential fuzzing against a reference TypeScript implementation

Using the `ffi` cheatcode, Murky tests its own Merkle root implementation against a TypeScript implementation using data provided by Forge's fuzzer:

```
function testMerkleRootMatchesJSImplementationFuzzed(bytes32[] memory leaves)
public {
    vm.assume(leaves.length > 1);
    bytes memory packed = abi.encodePacked(leaves);
    string[] memory runJsInputs = new string[](8);

    // Build ffi command string
    runJsInputs[0] = 'npm';
    runJsInputs[1] = '--prefix';
    runJsInputs[2] = 'differential_testing/scripts/';
    runJsInputs[3] = '--silent';
    runJsInputs[4] = 'run';
    runJsInputs[5] = 'generate-root-cli';
    runJsInputs[6] = leaves.length.toString();
    runJsInputs[7] = packed.toHexString();

    // Run command and capture output
    bytes memory jsResult = vm.ffi(runJsInputs);
    bytes32 jsGeneratedRoot = abi.decode(jsResult, (bytes32));

    // Calculate root using Murky
    bytes32 murkyGeneratedRoot = m.getRoot(leaves);
    assertEq(murkyGeneratedRoot, jsGeneratedRoot);
}
```

Note: see `Strings2.sol` in the Murky Repo for the library that enables `(bytes memory).toHexString()`

Forge runs `npm --prefix differential_testing/scripts/ --silent run generate-root-cli {numLeaves} {hexEncodedLeaves}`. This calculates the Merkle root for the input data using the reference JavaScript implementation. The script prints the root to stdout, and that printout is captured as `bytes` in the return value of `vm.ffi()`.

The test then calculates the root using the Solidity implementation.

Finally, the test asserts that the both roots are exactly equal. If they are not equal, the test fails.

Differential fuzzing against OpenZeppelin's Merkle Proof Library

You may want to use differential testing against another Solidity implementation. In that case, `ffi` is not needed. Instead, the reference implementation is imported directly into the test.

```
import "openzeppelin-contracts/contracts/utils/cryptography/MerkleProof.sol";
//...
function testCompatibilityOpenZeppelinProver(bytes32[] memory _data, uint256
node) public {
    vm.assume(_data.length > 1);
    vm.assume(node < _data.length);
    bytes32 root = m.getRoot(_data);
    bytes32[] memory proof = m.getProof(_data, node);
    bytes32 valueToProve = _data[node];
    bool murkyVerified = m.verifyProof(root, proof, valueToProve);
    bool ozVerified = MerkleProof.verify(proof, root, valueToProve);
    assertTrue(murkyVerified == ozVerified);
}
```

Differential testing against a known edge case

Differential tests are not always fuzzed – they are also useful for testing known edge cases. In the case of the Murky codebase, the initial implementation of the `log2ceil` function did not work for certain arrays whose lengths were close to a power of 2 (like 129). As a safety check, a test is always run against an array of this length and compared to the TypeScript implementation. You can see the full test here.

Standardized Testing against reference data

FFI is also useful for injecting reproducible, standardized data into the testing environment. In the Murky library, this is used as a benchmark for gas snapshotting (see forge snapshot).

```
bytes32[100] data;
uint256[8] leaves = [4, 8, 15, 16, 23, 42, 69, 88];

function setUp() public {
    string[] memory inputs = new string[](2);
    inputs[0] = "cat";
    inputs[1] = "src/test/standard_data/StandardInput.txt";
    bytes memory result =  vm.ffi(inputs);
    data = abi.decode(result, (bytes32[100]));
    m = new Merkle();
}

function testMerkleGenerateProofStandard() public view {
    bytes32[] memory _data = _getData();
    for (uint i = 0; i < leaves.length; ++i) {
        m.getProof(_data, leaves[i]);
    }
}
```

`src/test/standard_data/StandardInput.txt` is a text file that contains an encoded `bytes32[100]` array. It's generated outside of the test and can be used in any language's Web3 SDK. It looks something like:

```
0xf910ccaa307836354233316666386231414464306335333243453944383735313..423532
```

The standardized testing contract reads in the file using `ffi`. It decodes the data into an array and then, in this example, generates proofs for 8 different leaves. Because the data is constant and standard, we can meaningfully measure gas and performance improvements using this test.

> Of course, one could just hardcode the array into the test! But that makes it much harder to do consistent testing across contracts, implementations, etc.

Example: Differential Testing Gradual Dutch Auctions

The reference implementation for Paradigm's Gradual Dutch Auction mechanism contains a number of differential, fuzzed tests. It is an excellent repository to further explore differential testing using `ffi`.

- Differential tests for Discrete GDAs
- Differential tests for Continuous GDAs
- Reference Python implementation

Reference Repositories

- Gradual Dutch Auctions
- Murky
- Solidity Fuzzing Template

If you have another repository that would serve as a reference, please contribute it!

Deploying

Forge can deploy smart contracts to a given network with the `forge create` command.

Forge can deploy only one contract at a time.

To deploy a contract, you must provide a RPC URL (env: `ETH_RPC_URL`) and the private key of the account that will deploy the contract.

To deploy `MyContract` to a network:

```
$ forge create --rpc-url <your_rpc_url> --private-key <your_private_key>
src/MyContract.sol:MyContract
compiling...
success.
Deployer: 0xa735b3c25f...
Deployed to: 0x4054415432...
Transaction hash: 0x6b4e0ff93a...
```

Solidity files may contain multiple contracts. `:MyContract` above specifies which contract to deploy from the `src/MyContract.sol` file.

Use the `--constructor-args` flag to pass arguments to the constructor:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

import {ERC20} from "solmate/tokens/ERC20.sol";

contract MyToken is ERC20 {
    constructor(
        string memory name,
        string memory symbol,
        uint8 decimals,
        uint256 initialSupply
    ) ERC20(name, symbol, decimals) {
        _mint(msg.sender, initialSupply);
    }
}
```

Additionally, we can tell Forge to verify our contract on Etherscan, Sourcify or Blockscout, if the network is supported, by passing `--verify` .

```
$ forge create --rpc-url <your_rpc_url> \
    --constructor-args "ForgeUSD" "FUSD" 18 1000000000000000000000 \
    --private-key <your_private_key> \
    --etherscan-api-key <your_etherscan_api_key> \
    --verify \
    src/MyToken.sol:MyToken
```

Verifying a pre-existing contract

It is recommended to use the `--verify` flag with `forge create` to automatically verify the contract on explorer after a deployment. Note that for Etherscan `ETHERSCAN_API_KEY` must be set.

If you are verifying an already deployed contract, read on.

You can verify a contract on Etherscan, Sourcify or Blockscout with the `forge verify-contract` command.

You must provide:

- the contract address
- the contract name or the path to the contract `<path>:<contractname>`
- your Etherscan API key (env: `ETHERSCAN_API_KEY`) (if verifying on Etherscan).

Moreover, you may need to provide:

- the constructor arguments in the ABI-encoded format, if there are any
- compiler version used for build, with 8 hex digits from the commit version prefix (the commit will usually not be a nightly build). It is auto-detected if not specified.
- the number of optimizations, if the Solidity optimizer was activated. It is auto-detected if not specified.
- the chain ID, if the contract is not on Ethereum Mainnet

Let's say you want to verify `MyToken` (see above). You set the number of optimizations to 1 million, compiled it with v0.8.10, and deployed it, as shown above, to the Sepolia testnet (chain ID: 11155111). Note that `--num-of-optimizations` will default to 0 if not set on verification, while it defaults to 200 if not set on deployment, so make sure you pass `--num-of-optimizations 200` if you left the default compilation settings.

Here's how to verify it:

```
forge verify-contract \
    --chain-id 11155111 \
    --num-of-optimizations 1000000 \
    --watch \
    --constructor-args $(cast abi-encode
"constructor(string,string,uint256,uint256)" "ForgeUSD" "FUSD" 18
1000000000000000000000) \
    --etherscan-api-key <your_etherscan_api_key> \
    --compiler-version v0.8.10+commit.fc410830 \
    <the_contract_address> \
    src/MyToken.sol:MyToken

Submitted contract for verification:
                Response: `OK`
                GUID: `a6yrbjp5prvakia6bqp5qdacczyfhkyi5j1r6qbds1js41ak1a`
                url: https://sepolia.etherscan.io//address/0x6a54…3a4c#code
```

It is recommended to use the `--watch` flag along with `verify-contract` command in order to poll for the verification result.

If the `--watch` flag was not supplied, you can check the verification status with the `forge verify-check` command:

```
$ forge verify-check --chain-id 11155111 <GUID> <your_etherscan_api_key>
Contract successfully verified.
```

💡 **Tip**

Use Cast's `abi-encode` to ABI-encode arguments.

In this example, we ran `cast abi-encode "constructor(string,string,uint8,uint256)" "ForgeUSD" "FUSD" 18 1000000000000000000000` to ABI-encode the arguments.

Troubleshooting

`Invalid character 'x' at position 1`

Make sure the private key string does not begin with `0x`.

`EIP-1559 not activated`

EIP-1559 is not supported or not activated on the RPC server. Pass the `--legacy` flag to use legacy transactions instead of the EIP-1559 ones. If you do development in a local environment, you can use Hardhat instead of Ganache.

`Failed to parse tokens`

Make sure the passed arguments are of correct type.

`Signature error`

Make sure the private key is correct.

`Compiler version commit for verify`

If you want to check the exact commit you are running locally, try: `~/.svm/0.x.y/solc-0.x.y --version` where `x` and `y` are major and minor version numbers respectively. The output of this will be something like:

```
solc, the solidity compiler commandline interface
Version: 0.8.12+commit.f00d7308.Darwin.appleclang
```

Note: You cannot just paste the entire string "0.8.12+commit.f00d7308.Darwin.appleclang" as the argument for the compiler-version. But you can use the 8 hex digits of the commit to look up exactly what you should copy and paste from compiler version.

Known Issues

Verifying Contracts With Ambiguous Import Paths

Forge passes source directories (`src` , `lib` , `test` etc) as `--include-path` arguments to the compiler. This means that given the following project tree

```
|- src
|-- folder
|--- Contract.sol
|--- IContract.sol
```

it is possible to import `IContract` inside the `Contract.sol` using `folder/IContract.sol` import path.

Etherscan is not able to recompile such sources. Consider changing the imports to use relative import path.

Verifying Contracts With No Bytecode Hash

Currently, it's not possible to verify contracts on Etherscan with `bytecode_hash` set to `none`. Click here to learn more about how metadata hash is used for source code verification.

Gas Tracking

Forge can help you estimate how much gas your contract will consume.

Currently, Forge ships with two different tools for this job, but they may be merged in the future:

- **Gas reports**: Gas reports give you an overview of how much Forge thinks the individual functions in your contracts will consume in gas.
- **Gas snapshots**: Gas snapshots give you an overview of how much each test consumes in gas.

Gas reports and gas snapshots differ in some ways:

- Gas reports use tracing to figure out gas costs for individual contract calls. This gives more granular insight, at the cost of speed.
- Gas snapshots have more built-in tools, such as diffs and exporting the results to a file. Snapshots are not as granular as gas reports, but they are faster to generate.

Gas Reports

Forge can produce gas reports for your contracts. You can configure which contracts output gas reports via the `gas_reports` field in `foundry.toml`.

To produce reports for specific contracts:

```
gas_reports = ["MyContract", "MyContractFactory"]
```

To produce reports for all contracts:

```
gas_reports = ["*"]
```

To generate gas reports, run `forge test --gas-report`.

You can also use it in combination with other subcommands, such as `forge test --match-test testBurn --gas-report`, to generate only a gas report relevant to this test.

Example output:

MockERC1155 contract					
Deployment Cost	Deployment Size				
1082720	5440				
Function Name	min	avg	median	max	# calls
balanceOf	596	596	596	596	44
balanceOfBatch	2363	4005	4005	5647	2
batchBurn	2126	5560	2584	11970	3
batchMint	2444	135299	125081	438531	18
burn	814	2117	2117	3421	2
isApprovedForAll	749	749	749	749	1
mint	26039	31943	27685	118859	22
safeBatchTransferFrom	2561	137750	126910	461304	8
safeTransferFrom	1335	34505	28103	139557	9
setApprovalForAll	24485	24485	24485	24485	12

```
╭──────────────────────────┬──────────────────┬────────┬────────┬────────┬──────────╮
│ Example contract         ┆                  ┆        ┆        ┆        ┆          │
╞══════════════════════════╪══════════════════╪========╪========╪========╪==========╡
│ Deployment Cost          ┆ Deployment Size  ┆        ┆        ┆        ┆          │
┼┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┼┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┼┄┄┄┄┄┄┄┄┼┄┄┄┄┄┄┄┄┼┄┄┄┄┄┄┄┄┼┄┄┄┄┄┄┄┄┼┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄┄
```

MockERC1155 contract					
Deployment Cost	Deployment Size				
1082720	5440				
Function Name	min	avg	median	max	# calls
balanceOf	596	596	596	596	44
balanceOfBatch	2363	4005	4005	5647	2
batchBurn	2126	5560	2584	11970	3
batchMint	2444	135299	125081	438531	18
burn	814	2117	2117	3421	2
isApprovedForAll	749	749	749	749	1
mint	26039	31943	27685	118859	22
safeBatchTransferFrom	2561	137750	126910	461304	8
safeTransferFrom	1335	34505	28103	139557	9
setApprovalForAll	24485	24485	24485	24485	12

Gas Snapshots

Forge can generate gas snapshots for all your test functions. This can be useful to get a general feel for how much gas your contract will consume, or to compare gas usage before and after various optimizations.

To generate the gas snapshot, run `forge snapshot`.

This will generate a file called `.gas-snapshot` by default with all your tests and their respective gas usage.

```
$ forge snapshot
$ cat .gas-snapshot

ERC20Test:testApprove() (gas: 31162)
ERC20Test:testBurn() (gas: 59875)
ERC20Test:testFailTransferFromInsufficientAllowance() (gas: 81034)
ERC20Test:testFailTransferFromInsufficientBalance() (gas: 81662)
ERC20Test:testFailTransferInsufficientBalance() (gas: 52882)
ERC20Test:testInfiniteApproveTransferFrom() (gas: 90167)
ERC20Test:testMetadata() (gas: 14606)
ERC20Test:testMint() (gas: 53830)
ERC20Test:testTransfer() (gas: 60473)
ERC20Test:testTransferFrom() (gas: 84152)
```

Filtering

If you would like to specify a different output file, run `forge snapshot --snap <FILE_NAME>`.

You can also sort the results by gas usage. Use the `--asc` option to sort the results in ascending order and `--desc` to sort the results in descending order.

Finally, you can also specify a min/max gas threshold for all your tests. To only include results above a threshold, you can use the `--min <VALUE>` option. In the same way, to only include results under a threshold, you can use the `--max <VALUE>` option.

Keep in mind that the changes will be made in the snapshot file, and not in the snapshot being displayed on your screen.

You can also use it in combination with the filters for `forge test`, such as `forge snapshot --match-path contracts/test/ERC721.t.sol` to generate a gas snapshot relevant to this test contract.

Comparing gas usage

If you would like to compare the current snapshot file with your latest changes, you can use the `--diff` or `--check` options.

`--diff` will compare against the snapshot and display changes from the snapshot.

It can also optionally take a file name (`--diff <FILE_NAME>`), with the default being `.gas-snapshot` .

For example:

```
$ forge snapshot --diff .gas-snapshot2

Running 10 tests for src/test/ERC20.t.sol:ERC20Test
[PASS] testApprove() (gas: 31162)
[PASS] testBurn() (gas: 59875)
[PASS] testFailTransferFromInsufficientAllowance() (gas: 81034)
[PASS] testFailTransferFromInsufficientBalance() (gas: 81662)
[PASS] testFailTransferInsufficientBalance() (gas: 52882)
[PASS] testInfiniteApproveTransferFrom() (gas: 90167)
[PASS] testMetadata() (gas: 14606)
[PASS] testMint() (gas: 53830)
[PASS] testTransfer() (gas: 60473)
[PASS] testTransferFrom() (gas: 84152)
Test result: ok. 10 passed; 0 failed; finished in 2.86ms
testBurn() (gas: 0 (0.000%))
testFailTransferFromInsufficientAllowance() (gas: 0 (0.000%))
testFailTransferFromInsufficientBalance() (gas: 0 (0.000%))
testFailTransferInsufficientBalance() (gas: 0 (0.000%))
testInfiniteApproveTransferFrom() (gas: 0 (0.000%))
testMetadata() (gas: 0 (0.000%))
testMint() (gas: 0 (0.000%))
testTransfer() (gas: 0 (0.000%))
testTransferFrom() (gas: 0 (0.000%))
testApprove() (gas: -8 (-0.000%))
Overall gas change: -8 (-0.000%)
```

`--check` will compare a snapshot with an existing snapshot file and display all the differences, if any. You can change the file to compare against by providing a different file name: `--check <FILE_NAME>` .

For example:

```
$ forge snapshot --check .gas-snapshot2

Running 10 tests for src/test/ERC20.t.sol:ERC20Test
[PASS] testApprove() (gas: 31162)
[PASS] testBurn() (gas: 59875)
[PASS] testFailTransferFromInsufficientAllowance() (gas: 81034)
[PASS] testFailTransferFromInsufficientBalance() (gas: 81662)
[PASS] testFailTransferInsufficientBalance() (gas: 52882)
[PASS] testInfiniteApproveTransferFrom() (gas: 90167)
[PASS] testMetadata() (gas: 14606)
[PASS] testMint() (gas: 53830)
[PASS] testTransfer() (gas: 60473)
[PASS] testTransferFrom() (gas: 84152)
Test result: ok. 10 passed; 0 failed; finished in 2.47ms
Diff in "ERC20Test::testApprove()": consumed "(gas: 31162)" gas, expected "
(gas: 31170)" gas
```

Debugger

Forge ships with an interactive debugger.

The debugger is accessible on `forge debug` and on `forge test`.

Using `forge test`:

```
$ forge test --debug $FUNC
```

Where `$FUNC` is the signature of the function you want to debug. For example:

```
$ forge test --debug "testSomething()"
```

If you have multiple contracts with the same function name, you need to limit the matching functions down to only one case using `--match-path` and `--match-contract`.

If the matching test is a fuzz test, the debugger will open the first failing fuzz scenario, or the last successful one, whichever comes first.

Using `forge debug`:

```
$ forge debug --debug $FILE --sig $FUNC
```

Where `$FILE` is the path to the contract you want to debug, and `$FUNC` is the signature of the function you want to debug. For example:

```
$ forge debug --debug src/SomeContract.sol --sig "myFunc(uint256,string)" 123
"hello"
```

You can also specify raw calldata using `--sig` instead of a function signature.

If your source file contains more than one contract, specify the contract you want to debug using the `--target-contract` flag.

Debugger layout

When the debugger is run, you are presented with a terminal divided into four quadrants:

- **Quadrant 1**: The opcodes in the debugging session, with the current opcode highlighted. Additionally, the address of the current account, the program counter and the accumulated gas usage is also displayed
- **Quadrant 2**: The current stack, as well as the size of the stack
- **Quadrant 3**: The source view
- **Quadrant 4**: The current memory of the EVM

As you step through your code, you will notice that the words in the stack and memory sometimes change color.

For the memory:

- **Red words** are about to be written to by the current opcode
- **Green words** were written to by the previous opcode
- **Cyan words** are being read by the current opcode

For the stack, **cyan words** are either being read or popped by the current opcode.

⚠️ Note

In most test frameworks, the first test assertion to fail is the one reported. In foundry, the last test assertion to fail (that comes from DSTest or cheatcodes) is the one to be reported.

Navigating

General

- `q`: Quit the debugger
- `h`: Show help

Navigating calls

- `0-9` + `k`: Step a number of times backwards (alternatively scroll up with your mouse)
- `0-9` + `j`: Step a number of times forwards (alternatively scroll down with your mouse)
- `g`: Move to the beginning of the transaction
- `G`: Move to the end of the transaction
- `c`: Move to the previous call-type instruction (i.e. `CALL`, `STATICCALL`, `DELEGATECALL`, and `CALLCODE`).
- `C`: Move to the next call-type instruction
- `a`: Move to the previous `JUMP` or `JUMPI` instruction
- `s`: Move to the next `JUMPDEST` instruction
- `'` + `a-z`: Move to `<char>` breakpoint set by a `vm.breakpoint` cheatcode

Navigating memory

- `Ctrl` + `j`: Scroll the memory view down
- `Ctrl` + `k`: Scroll the memory view up
- `m`: Show memory as UTF8

Navigating the stack

- `J`: Scroll the stack view down
- `K`: Scroll the stack view up
- `t`: Show labels on the stack to see what items the current op will consume

Overview of Cast

Cast is Foundry's command-line tool for performing Ethereum RPC calls. You can make smart contract calls, send transactions, or retrieve any type of chain data - all from your command-line!

How to use Cast

To use Cast, run the `cast` command followed by a subcommand:

```
$ cast <subcommand>
```

Examples

Let's use `cast` to retrieve the total supply of the DAI token:

```
$ cast call 0x6b175474e89094c44da98b954eedeac495271d0f "totalSupply()(uint256)"
--rpc-url https://eth-mainnet.alchemyapi.io/v2/Lc7oIGYeL_QvInzI0Wiu_pOZZDEKBrdf
3245167686810814481267213974 [3.245e27]
```

`cast` also provides many convenient subcommands, such as for decoding calldata:

```
$ cast 4byte-decode
0x1F1F897F676d000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000003e7
1) "fulfillRandomness(bytes32,uint256)"
0x676d000000000000000000000000000000000000000000000000000000000000000000000000
999
```

You can also use `cast` to send arbitrary messages. Here's an example of sending a message between two Anvil accounts.

```
$ cast send --private-key <Your Private Key>
0x3c44cdddb6a900fa2b585dd299e03d12fa4293bc $(cast from-utf8 "hello world") --
rpc-url http://127.0.0.1:8545/
```

📚 **Reference**

See the `cast` Reference for a complete overview of all the available subcommands.

Overview of Anvil

Anvil is a local testnet node shipped with Foundry. You can use it for testing your contracts from frontends or for interacting over RPC.

Anvil is part of the Foundry suite and is installed alongside `forge`, `cast`, and `chisel`. If you haven't installed Foundry yet, see Foundry installation.

Note: If you have an older version of Foundry installed, you'll need to re-install `foundryup` in order for Anvil to be downloaded.

How to use Anvil

To use Anvil, simply type `anvil`. You should see a list of accounts and private keys available for use, as well as the address and port that the node is listening on.

Anvil is highly configurable. You can run `anvil -h` to see all the configuration options.

Some basic options are:

```
#  Number of dev accounts to generate and configure. [default: 10]
anvil -a, --accounts <ACCOUNTS>

# The EVM hardfork to use. [default: latest]
anvil --hardfork <HARDFORK>

# Port number to listen on. [default: 8545]
anvil  -p, --port <PORT>
```

📚 **Reference**

See the `anvil` Reference for in depth information on Anvil and its capabilities.

Overview of Chisel

Chisel is an advanced Solidity REPL shipped with Foundry. It can be used to quickly test the behavior of Solidity snippets on a local or forked network.

Chisel is part of the Foundry suite and is installed alongside `forge`, `cast`, and `anvil`. If you haven't installed Foundry yet, see Foundry installation.

> Note: If you have an older version of Foundry installed, you'll need to re-install `foundryup` in order for Chisel to be downloaded.

How to use Chisel

To use Chisel, simply type `chisel`. From there, start writing Solidity code! Chisel will offer verbose feedback on each input.

Chisel can be used both within and outside of a foundry project. If the binary is executed in a Foundry project root, Chisel will inherit the project's configuration options.

To see available commands, type `!help` within the REPL.

> 📚 **Reference**
>
> See the `chisel` Reference for in depth information on Chisel and its capabilities.

Configuring with `foundry.toml`

Forge can be configured using a configuration file called `foundry.toml`, which is placed in the root of your project.

Configuration can be namespaced by profiles. The default profile is named `default`, from which all other profiles inherit. You are free to customize the `default` profile, and add as many new profiles as you need.

Additionally, you can create a global `foundry.toml` in your home directory.

Let's take a look at a configuration file that contains two profiles: the default profile, which always enables the optimizer, as well as a CI profile, that always displays traces:

```
[profile.default]
optimizer = true
optimizer_runs = 20_000

[profile.ci]
verbosity = 4
```

When running `forge`, you can specify the profile to use using the `FOUNDRY_PROFILE` environment variable.

Standalone sections

Besides the profile sections, the configuration file can also contain standalone sections (`[fmt]`, `[fuzz]`, `[invariant]` etc). By default, each standalone section belongs to the `default` profile. i.e. `[fmt]` is equivalent to `[profile.default.fmt]`.

To configure the standalone section for different profiles other than `default`, use syntax `[profile.<profile name>.<standalone>]`. i.e. `[profile.ci.fuzz]`.

📚 **Reference**

See the `foundry.toml` Reference for a complete overview of what you can configure.

Continuous Integration

GitHub Actions

To test your project using GitHub Actions, here is a sample workflow:

```yaml
on: [push]

name: test

jobs:
  check:
    name: Foundry project
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
        with:
          submodules: recursive

      - name: Install Foundry
        uses: foundry-rs/foundry-toolchain@v1
        with:
          version: nightly

      - name: Run tests
        run: forge test -vvv
```

Travis CI

To test your project using Travis CI, here is a sample workflow:

```yaml
language: rust
cache:
  cargo: true
  directories:
    - $HOME/.foundry

install:
  - curl -L https://foundry.paradigm.xyz | bash
  - export PATH=$PATH:$HOME/.foundry/bin
  - foundryup -b master

script:
  - forge test -vvv
```

GitLab CI

To test your project using GitLab CI, here is a sample workflow: Note: check out Policy to fetch the remote image

```
variables:
  GIT_SUBMODULE_STRATEGY: recursive

jobs:
  image: ghcr.io/foundry-rs/foundry
  script:
    - forge install
    - forge test -vvv
```

Integrating with VSCode

You can get Solidity support for Visual Studio Code by installing the VSCode Solidity extension.

To make the extension play nicely with Foundry, you may have to tweak a couple of things.

1. Remappings

You may want to place your remappings in `remappings.txt`.

If they are already in `foundry.toml`, copy them over and use `remappings.txt` instead. If you just use the autogenerated remappings that Foundry provides, run `forge remappings > remappings.txt`.

2. Dependencies

You may have to add the following to your `.vscode/settings.json` for the extension to find your dependencies:

```
{
  "solidity.packageDefaultDependenciesContractsDirectory": "src",
  "solidity.packageDefaultDependenciesDirectory": "lib"
}
```

Where `src` is the source code directory and `lib` is your dependency directory.

3. Formatter

To enable the built-in formatter that comes with Foundry to automatically format your code on save, you can add the following settings to your `.vscode/settings.json`:

```
{
  "editor.formatOnSave": true,
  "[solidity]": {
    "editor.defaultFormatter": "JuanBlanco.solidity"
  },
  "solidity.formatter": "forge",
}
```

To configure the formatter settings, refer to the Formatter reference.

4. Solc Version

Finally, it is recommended to specify a Solidity compiler version:

```
"solidity.compileUsingRemoteVersion": "v0.8.17"
```

To get Foundry in line with the chosen version, add the following to your `default` profile in `foundry.toml`.

```
solc = "0.8.17"
```

Shell Autocompletion

You can generate autocompletion shell scripts for `bash`, `elvish`, `fish`, `powershell`, and `zsh`.

zsh

First, ensure that the following is present somewhere in your `~/.zshrc` file (if not, add it):

```
autoload -U compinit
compinit -i
```

Then run:

```
forge completions zsh | sudo tee /usr/local/share/zsh/site-functions/_forge
cast completions zsh | sudo tee /usr/local/share/zsh/site-functions/_cast
anvil completions zsh | sudo tee /usr/local/share/zsh/site-functions/_anvil
```

For macOS:

```
forge completions zsh > /opt/homebrew/completions/zsh/_forge
cast completions zsh > /opt/homebrew/completions/zsh/_cast
anvil completions zsh > /opt/homebrew/completions/zsh/_anvil
```

fish

```
mkdir -p $HOME/.config/fish/completions
forge completions fish > $HOME/.config/fish/completions/forge.fish
cast completions fish > $HOME/.config/fish/completions/cast.fish
anvil completions fish > $HOME/.config/fish/completions/anvil.fish
source $HOME/.config/fish/config.fish
```

bash

```
mkdir -p $HOME/.local/share/bash-completion/completions
forge completions bash > $HOME/.local/share/bash-completion/completions/forge
cast completions bash > $HOME/.local/share/bash-completion/completions/cast
anvil completions bash > $HOME/.local/share/bash-completion/completions/anvil
exec bash
```

Static Analyzers

Slither

To test your project using slither, here is a sample `slither.config.json`:

```
{
  "filter_paths": "lib"
}
```

To run Slither on the entire project, use this command in the root of the project:

```
slither .
```

By default (as of version 0.10.0), this will skip tests and scripts. To force inclusion of the tests and scripts, add the `--foundry-compile-all` flag.

To run Slither on a single file, use this command:

```
slither src/Contract.sol
```

Note, this requires configuring the solc version in the foundry config file.

You do not need to provide remappings via the `solc_remaps` option as Slither will automatically detect remappings in a Foundry project. Slither will invoke `forge` to perform the build.

See the Slither wiki for more information.

In order to use a custom configuration, such as the sample `slither.config.json` mentioned above, the following command is used as mentioned in the slither-wiki. By default slither looks for the `slither.config.json` but you can define the path and any other `json` file of your choice:

```
slither --config-file <path>/file.config.json .
```

Example output (Raw):

```
Pragma version^0.8.13 (Counter.sol#2) necessitates a version too recent to be
trusted. Consider deploying with 0.6.12/0.7.6/0.8.7
solc-0.8.13 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#incorrect-versions-of-solidity

setNumber(uint256) should be declared external:
        - Counter.setNumber(uint256) (Counter.sol#7-9)
increment() should be declared external:
        - Counter.increment() (Counter.sol#11-13)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#public-function-that-could-be-declared-external
Counter.sol analyzed (1 contracts with 78 detectors), 4 result(s) found
```

Slither also has a GitHub Action for CI/CD.

Mythril

To test your project using mythril, here is a sample `mythril.config.json`:

```
{
  "remappings": [
    "ds-test/=lib/ds-test/src/",
    "forge-std/=lib/forge-std/src/"
  ],
  "optimizer": {
    "enabled": true,
    "runs": 200
  }
}
```

Note, you need switch `rustc` to nightly to install `mythril`:

```
rustup default nightly
pip3 install mythril
myth analyze src/Contract.sol --solc-json mythril.config.json
```

See the mythril docs for more information.

You can pass custom Solc compiler output to Mythril using the `--solc-json` flag. For example:

```
$ myth analyze src/Counter.sol --solc-json mythril.config.json
.
.
mythril.laser.plugin.loader [INFO]: Loading laser plugin: coverage
mythril.laser.plugin.loader [INFO]: Loading laser plugin: mutation-pruner
.
.
Achieved 11.56% coverage for code:
6080604052348015610010576000 80fd5b5060f78061001f6000396000f3fe60806040523480156
00f57600080fd5b5060043610603c5760003560e01c80633fb5c1cb1460415780638381f58a1460
53578063d09de08a14606d575b600080fd5b6051604c3660046083565b600055565b005b605b600
05481565b604051908152602001604051809 10390f35b60516000805490806 07c83609b565b9190
505550565b6000602082840312156094576000 80fd5b5035919050565b60006001820160ba57634
e487b7160e01b600052601160045260246000fd5b506001019056fea2646970667358221220659f
ce8aadca285da9206b61f95de294d3958c409cc3011ded856f421885867464736f6c63430008100
033
mythril.laser.plugin.plugins.coverage.coverage_plugin [INFO]: Achieved 90.13%
coverage for code:
6080604052348015600f57600080fd5b5060043610603c5760003560e01c80633fb5c1cb1460415
780638381f58a146053578063d09de08a14606d575b600080fd5b6051604c3660046083565b6000
55565b005b605b60005481565b604051908152602001604051809 10390f35b60516000805490806
07c83609b565b9190505550565b6000602082840312156094576000 80fd5b5035919050565b6000
6001820160ba57634e487b7160e01b600052601160045260246000fd5b506001019056fea264697
0667358221220659fce8aadca285da9206b61f95de294d3958c409cc3011ded856f421885867464
736f6c63430008100033
mythril.laser.plugin.plugins.instruction_profiler [INFO]: Total:
1.0892839431762695 s
[ADD         ]   0.9974 %,  nr      9,  total   0.0109 s,  avg   0.0012 s,  min
0.0011 s,  max   0.0013 s
.
.
[SWAP1       ]   1.8446 %,  nr     18,  total   0.0201 s,  avg   0.0011 s,  min
0.0010 s,  max   0.0013 s
[SWAP2       ]   0.8858 %,  nr      9,  total   0.0096 s,  avg   0.0011 s,  min
0.0010 s,  max   0.0011 s

mythril.analysis.security [INFO]: Starting analysis
mythril.mythril.mythril_analyzer [INFO]: Solver statistics:
Query count: 61
Solver time: 3.6820807456970215
The analysis was completed successfully. No issues were detected.
```

The findings will be listed at the end of this output if any. Since the default `Counter.sol` doesn't have any logic, `mythx` reports that no issues were found.

Integrating with Hardhat

It's possible to have your Foundry project work alongside Hardhat. This article assumes that you have Foundry and node installed in your system. This article also assumes familiarity with both Foundry and Hardhat.

Why does this not work out of the box?

Hardhat by default expects libraries to be installed in `node_modules`, the default folder for all NodeJS dependencies. Foundry expects them to be in `lib`. Of course we can configure Foundry but not easily to the directory structure of `node_modules`.

For this reason, the recommended setup is to use hardhat-foundry. When hardhat-foundry is installed and used correctly, Hardhat will use the same contracts directory that is used by Foundry, and it will be able to use dependencies installed with forge install.

In this article we will cover both scenarios:

1. Adding Hardhat to a Foundry project, and,
2. Adding Foundry to a Hardhat project.

Just show me the example repo!

Enjoy!

If you want to adapt this to a Foundry project you already have or learn how it works, read below:

Adding Hardhat to a Foundry project

Inside your Foundry project working directory:

1. `npm init -y` - This will set up a `package.json` file.
2. `npm i --save-dev hardhat` - Install Hardhat as a dev dependency into your project.
3. `npx hardhat init` - Initialize your Hardhat project inside the same directory and choose the "**Create an empty hardhat.config.js**" option. This will create a basic `hardhat.config.js` file.
4. `npm i --save-dev @nomicfoundation/hardhat-foundry @nomicfoundation/hardhat-toolbox` - This will install the hardhat-foundry plugin and the Hardhat toolbox plugin which is a combination of all the basic dependencies you need to run Hardhat tests.

Your hardhat.config.js file should look like this to make the plugins work:

```
require("@nomicfoundation/hardhat-toolbox");
require("@nomicfoundation/hardhat-foundry");
/** @type import('hardhat/config').HardhatUserConfig */
module.exports = {
  solidity: "0.8.19",
};
```

5. By default, a Foundry project ships with a simple `Counter.sol` contract and a couple of tests. Create a file named `Counter.t.js` inside the `test` directory parallel to the default `Counter.t.sol` file.
6. Add the following code to the `Counter.t.js` file:

```
const { expect } = require("chai");
const hre = require("hardhat");
const { loadFixture } = require("@nomicfoundation/hardhat-toolbox/network-
helpers");

describe("Counter contract", function () {
  async function CounterLockFixture() {
    const counter = await ethers.deployContract("Counter");
    await counter.setNumber(0);

    return { counter };
  }

  it("Should increment the number correctly", async function () {
    const { counter } = await loadFixture(CounterLockFixture);
    await counter.increment();
    expect(await counter.number()).to.equal(1);
  });

  // This is not a fuzz test because Hardhat does not support fuzzing yet.
  it("Should set the number correctly", async function () {
    const { counter } = await loadFixture(CounterLockFixture);
    await counter.setNumber(100);
    expect(await counter.number()).to.equal(100);
  });
});
```

This piece of code will execute the same tests as the default `Counter.t.sol` file.

And this is it! You can create Hardhat and Foundry tests in the same `test` directory and run them with `npx hardhat test` and `forge test` respectively. Check out Hardhat's documentation to learn more.

Adding Foundry to a Hardhat project

Inside your Hardhat project working directory:

1. `npm i --save-dev @nomicfoundation/hardhat-foundry` - Install the hardhat-foundry plugin.
2. Add `require("@nomicfoundation/hardhat-foundry");` to the top of your `hardhat.config.js` file.

ℹ️ **Note** Step number 3 will only work if your directory is an initialized git repository. Run `git init` if you haven't already.

---

3. Run `npx hardhat init-foundry` in your terminal. This will generate a `foundry.toml` file based on your Hardhat project's existing configuration, and will install the `forge-std` library.

Hardhat will now set up a basic Foundry project inside the same directory with a few configurations inside the `foundry.toml` file to make sure that Foundry knows where to look for your contracts, tests and dependencies. You can always change these configurations later by editing the `foundry.toml` file.

Best Practices

This guide documents the suggested best practices when developing with Foundry. In general, it's recommended to handle as much as possible with `forge fmt`, and anything this doesn't handle is below.

- Best Practices
  - General Contract Guidance
  - Tests
    - General Test Guidance
    - Fork Tests
    - Test Harnesses
      - Internal Functions
      - Private Functions
      - Workaround Functions
    - Best practices
    - Taint Analysis
  - Scripts
    - Private Key Management
  - Comments
  - Resources

General Contract Guidance

1. Always use named import syntax, don't import full files. This restricts what is being imported to just the named items, not everything in the file. Importing full files can result in solc complaining about duplicate definitions and slither erroring, especially as repos grow and have more dependencies with overlapping names.
   - Good: `import {MyContract} from "src/MyContract.sol"` to only import `MyContract`.
   - Bad: `import "src/MyContract.sol"` imports everything in `MyContract.sol`. (Importing `forge-std/Test` or `Script` can be an exception here, so you get the console library, etc).
2. Note the tradeoffs between absolute and relative paths for imports (where absolute paths are relative to the repo root, e.g. `"src/interfaces/IERC20.sol"`), and choose the best approach for your project:
   - Absolute paths make it easier to see where files are from and reduce churn when moving files around.

    - Relative paths make it more likely your editor can provide features like linting and autocomplete, since editors/extensions may not understand your remappings.

3. If copying a library from a dependency (instead of importing it), use the `ignore = []` option in the config file to avoid formatting that file. This makes diffing it against the original simpler for reviewers and auditors.

4. Similarly, feel free to use the `// forgefmt: disable-*` comment directives to ignore lines/sections of code that look better with manual formatting. Supported values for `*` are:

    - `disable-line`
    - `disable-next-line`
    - `disable-next-item`
    - `disable-start`
    - `disable-end`

Additional best practices from samsczun's How Do You Even Write Secure Code Anyways talk:

- Use descriptive variable names.
- Limit the number of active variables.
- No redundant logic.
- Early exit as much as possible to reduce mental load when seeing the code.
- Related code should be placed near each other.
- Delete unused code.

Tests

General Test Guidance

1. For testing `MyContract.sol`, the test file should be `MyContract.t.sol`. For testing `MyScript.s.sol`, the test file should be `MyScript.t.sol`.

    - If the contract is big and you want to split it over multiple files, group them logically like `MyContract.owner.t.sol`, `MyContract.deposits.t.sol`, etc.

2. Never make assertions in the `setUp` function, instead use a dedicated test like `test_SetUpState()` if you need to ensure your `setUp` function does what you expected. More info on why in foundry-rs/foundry#1291

3. For unit tests, there are two major ways to organize the tests:

    1. Treat contracts as describe blocks:

    - `contract Add` holds all unit tests for the `add` method of `MyContract`.
    - `contract Supply` holds all tests for the `supply` method.
    - `contract Constructor` hold all tests for the constructor.
    - A benefit of this approach is that smaller contracts should compile faster than large ones, so this approach of many small contracts should save time as test suites get large.

  2. Have a Test contract per contract-under-test, with as many utilities and fixtures as you want:

    - `contract MyContractTest` holds all unit tests for `MyContract`.
    - `function test_add_AddsTwoNumbers()` lives within `MyContractTest` to test the `add` method.
    - `function test_supply_UsersCanSupplyTokens()` also lives within `MyContractTest` to test the `supply` method.
    - A benefit of this approach is that test output is grouped by contract-under-test, which makes it easier to quickly see where failures are.

4. Some general guidance for all tests:

  - Test contracts/functions should be written in the same order as the original functions in the contract-under-test.
  - All unit tests that test the same function should live serially in the test file.
  - Test contracts can inherit from any helper contracts you want. For example `contract MyContractTest` tests `MyContract`, but may inherit from forge-std's `Test`, as well as e.g. your own `TestUtilities` helper contract.

5. Integration tests should live in the same `test` directory, with a clear naming convention. These may be in dedicated files, or they may live next to related unit tests in existing test files.

6. Be consistent with test naming, as it's helpful for filtering tests (e.g. for gas reports you might want to filter out revert tests). When combining naming conventions, keep them alphabetical. Below is a sample of valid names. A comprehensive list of valid and invalid examples can be found here.

  - `test_Description` for standard tests.
  - `testFuzz_Description` for fuzz tests.
  - `test_Revert[If|When]_Condition` for tests expecting a revert.
  - `testFork_Description` for tests that fork from a network.
  - `testForkFuzz_Revert[If|When]_Condition` for a fuzz test that forks and expects a revert.

7. Name your constants and immutables using `ALL_CAPS_WITH_UNDERSCORES`, to make it easier to distinguish them from variables and functions.

8. When using assertions like `assertEq`, consider leveraging the last string param to make it easier to identify failures. These can be kept brief, or even just be numbers—they basically serve as a replacement for showing line numbers of the revert, e.g. `assertEq(x, y, "1")` or `assertEq(x, y, "sum1")`. *(Note: foundry-rs/foundry#2328 tracks integrating this natively).*

9. When testing events, prefer setting all `expectEmit` arguments to `true`, i.e. `vm.expectEmit(true, true, true, true)` or `vm.expectEmit()`. Benefits:

    - This ensures you test everything in your event.
    - If you add a topic (i.e. a new indexed parameter), it's now tested by default.
    - Even if you only have 1 topic, the extra `true` arguments don't hurt.

10. Remember to write invariant tests! For the assertion string, use a verbose english description of the invariant: `assertEq(x + y, z, "Invariant violated: the sum of x and y must always equal z")`. For more info on this, check out the Invariant Testing tutorial.

Fork Tests

1. Don't feel like you need to give forks tests special treatment, and use them liberally:

    - Mocks are *required* in closed-source web2 development—you have to mock API responses because the code for that API isn't open source so you cannot just run it locally. But for blockchains that's not true: any code you're interacting with that's already deployed can be locally executed and even modified for free. So why introduce the risk of a wrong mock if you don't need to?
    - A common reason to avoid fork tests and prefer mocks is that fork tests are slow. But this is not always true. By pinning to a block number, forge caches RPC responses so only the first run is slower, and subsequent runs are significantly faster. See this benchmark, where it took forge 7 minutes for the first run with a remote RPC, but only half a second once data was cached. Alchemy, Infura and Tenderly offer free archive data, so pinning to a block shouldn't be problematic.
    - Note that the foundry-toolchain GitHub Action will cache RPC responses in CI by default, and it will also update the cache when you update your fork tests.

2. Be careful with fuzz tests on a fork to avoid burning through RPC requests with non-deterministic fuzzing. If the input to your fork fuzz test is some parameter which is used in an RPC call to fetch data (e.g. querying the token balance of an address), each run of a fuzz test uses at least 1 RPC request, so you'll quickly hit rate limits or usage limits. Solutions to consider:

    - Replace multiple RPC calls with a single multicall.
    - Specify a fuzz/invariant seed: this makes sure each `forge test` invocation uses the same fuzz inputs. RPC results are cached locally, so you'll only query the node

the first time.
    - In CI, consider setting the fuzz seed using a computed environment variable so it changes every day or every week. This gives flexibility on the tradeoff between increasing randomness to find more bugs vs. using a seed to reduce RPC requests.
    - Structure your tests so the data you are fuzzing over is computed locally by your contract, and not data that is used in an RPC call (may or may not be feasible based on what you're doing).
    - Lastly, you can of course always run a local node or bump your RPC plan.

3. When writing fork tests, do not use the `--fork-url` flag. Instead, prefer the following approach for its improved flexibility:
    - Define `[rpc_endpoints]` in the `foundry.toml` config file and use the forking cheatcodes.
    - Access the RPC URL endpoint in your test with forge-std's `stdChains.ChainName.rpcUrl`. See the list of supported chains and expected config file aliases here.
    - Always pin to a block so tests are deterministic and RPC responses are cached.
    - More info on this fork test approach can be found here (this predates `StdChains` so that aspect is a bit out of date).

Test Harnesses

Internal Functions

To test `internal` functions, write a harness contract that inherits from the contract under test (CuT). Harness contracts that inherit from the CuT expose the `internal` functions as `external` ones.

Each `internal` function that is tested should be exposed via an external one with a name that follows the pattern `exposed_<function_name>`. For example:

```
// file: src/MyContract.sol
contract MyContract {
  function myInternalMethod() internal returns (uint) {
    return 42;
  }
}

// file: test/MyContract.t.sol
import {MyContract} from "src/MyContract.sol";

contract MyContractHarness is MyContract {
  // Deploy this contract then call this method to test `myInternalMethod`.
  function exposed_myInternalMethod() external returns (uint) {
    return myInternalMethod();
  }
}
```

Private Functions

Unfortunately there is currently no good way to unit test `private` methods since they cannot be accessed by any other contracts. Options include:

- Converting `private` functions to `internal`.
- Copy/pasting the logic into your test contract and writing a script that runs in CI check to ensure both functions are identical.

Workaround Functions

Harnesses can also be used to expose functionality or information otherwise unavailable in the original smart contract. The most straightforward example is when we want to test the length of a public array. The functions should follow the pattern: `workaround_<function_name>`, such as `workaround_queueLength()`.

Another use case for this is tracking data that you would not track in production to help test invariants. For example, you might store a list of all token holders to simplify validation of the invariant "sum of all balances must equal total supply". These are often known as "ghost variables". You can learn more about this in Rikard Hjort's Formal Methods for the Working DeFi Dev talk.

Best practices

Thanks to @samsczun's How Do You Even Write Secure Code Anyways talk for the tips in this section and the following section.

- Don't optimize for coverage, optimize for well thought-out tests.
- Write positive and negative unit tests.

    - Write *positive* unit tests for things that the code should handle. Validate *all* state that changes from these tests.
    - Write *negative* unit tests for things that the code should *not* handle. It's helpful to follow up (as an adjacent test) with the positive test and make the change that it needs to pass.
    - Each code path should have its own unit test.
- Write integration tests to test entire features.
- Write fork tests to verify the correct behavior with existing deployed contract.

Taint Analysis

When testing, you should prioritize functions that an attacker can affect, that means functions that accept some kind of user input. These are called *sources*.

Consider that input data as *tainted* until it has been checked by the code, at which point it's considered *clean*.

A *sink* is a part of the code where some important operation is happening. For example, in MakerDAO that would be `vat.sol`.

You should *ensure* that no *tainted* data ever reaches a *sink*. That means that all data that find themselves in the sink, should, at some point, have been checked by you. So, you need to define what the data *should* be and then make sure your checks *ensure* that the data will be how you expect it to be.

Scripts

1. Stick with `run` as the default function name for clarity.

2. Any methods that are not intended to be called directly in the script should be `internal` or `private`. Generally the only public method should be `run`, as it's easier to read/understand when each script file just does one thing.

3. Consider prefixing scripts with a number based on the order they're intended to be run over the protocol's lifecycle. For example, `01_Deploy.s.sol`, `02_TransferOwnership.s.sol`. This makes things more self-documenting. This may not always apply depending on your project.

4. Test your scripts.

    - Unit test them like you would test normal contracts, by writing tests that assert on the state changes made from running the script.
    - Write your deploy script and scaffold tests by running that script. Then, run all tests against the state resulting from your production deployment script. This is a

great way to gain confidence in a deploy script.
    - Within your script itself, use `require` statements (or the `if (condition) revert()` pattern if you prefer) to stop execution of your script if something is wrong. For example, `require(computedAddress == deployedAddress, "address mismatch")`. Using the `assertEq` helpers instead will not stop execution.

5. **Carefully audit which transactions are broadcast**. Transactions not broadcast are still executed in the context of a test, so missing broadcasts or extra broadcasts are easy sources of error in the previous step.

6. **Watch out for frontrunning**. Forge simulates your script, generates transaction data from the simulation results, then broadcasts the transactions. Make sure your script is robust against chain-state changing between the simulation and broadcast. A sample script vulnerable to this is below:

```
// Pseudo-code, may not compile.
contract VulnerableScript is Script {
   function run() public {
      vm.startBroadcast();

      // Transaction 1: Deploy a new Gnosis Safe with CREATE.
      // Because we're using CREATE instead of CREATE2, the address of the new
      // Safe is a function of the nonce of the gnosisSafeProxyFactory.
      address mySafe = gnosisSafeProxyFactory.createProxy(singleton, data);

      // Transaction 2: Send tokens to the new Safe.
      // We know the address of mySafe is a function of the nonce of the
      // gnosisSafeProxyFactory. If someone else deploys a Gnosis Safe between
      // the simulation and broadcast, the address of mySafe will be different,
      // and this script will send 1000 DAI to the other person's Safe. In this
      // case, we can protect ourselves from this by using CREATE2 instead of
      // CREATE, but every situation may have different solutions.
      dai.transfer(mySafe, 1000e18);

      vm.stopBroadcast();
   }
}
```

7. For scripts that read from JSON input files, put the input files in `script/input/<chainID>/<description>.json`. Then have `run(string memory input)` (or take multiple string inputs if you need to read from multiple files) as the script's signature, and use the below method to read the JSON file.

```
function readInput(string memory input) internal returns (string memory) {
  string memory inputDir = string.concat(vm.projectRoot(),
"/script/input/");
  string memory chainDir = string.concat(vm.toString(block.chainid), "/");
  string memory file = string.concat(input, ".json");
  return vm.readFile(string.concat(inputDir, chainDir, file));
}
```

Private Key Management

Script execution requires a private key to send transactions. This key controls all funds in the account, so it must be protected carefully. There are a few options for securely broadcasting transactions through a script:

1. **Use a hardware wallet.** Hardware wallets such as Ledger and Trezor store seed phrases in a secure enclave. Forge can send a raw transaction to the wallet, and the wallet will sign the transaction. The signed transaction is returned to forge and broadcaster. This way, private keys never leave the hardware wallet, making this a very secure approach. To use a hardware wallet with scripts, see the `--ledger` and `--trezor` flags.

2. **Use a private key directly.** With this approach you expose a private key on your machine, making it riskier than the above option. Therefore the suggested way to directly use a private key is to generate a new wallet for executing the script, and only send that wallet enough funds to run the script. Then, stop using the key after the script is complete. This way, if the key is compromised, only the funds on this throwaway key are lost, as opposed to losing everything in your wallet.

    1. With this approach, it's very important that your scripts or contracts don't rely on `msg.sender` since the sender will not be an account that's meant to be used again. For example, if a deploy script configures a contract owner, ensure the owner a constructor argument and not set to `msg.sender`.
    2. To use this approach, you can either store the private key in an environment variable and use cheat codes to read it in, or use the `--private-key` flag to directly provide the key.

3. **Use a keystore.** This can be thought of as a middle ground between the above two approaches. With `cast wallet import` you import a private key and encrypt it with a

password. This still temporarily exposes your private key on your machine, but it becomes encrypted and you'll provide the password to decrypt it to run a script.

Additional security precautions when using scripts:

1. Use a separate wallet for testing and development, instead of using your main wallet with real funds. Diversifying minimizes the risk of losing funds if your development wallet is compromised.
2. If you accidentally push a private key or seed phrase to GitHub, or expose it online via other means—even momentarily—treat it as compromised. Act immediately to transfer your funds to a safer destination.
3. When in doubt about whether a wallet contains real funds or not, assume it does. Always be certain about a wallet's balances and status when using it for development purposes. Use blockscan to easily check many chains to see where the address has been used.
4. Remember that adding an account in wallets like Metamask generates a new private key. However, that private key is derived from the same mnemonic as the other accounts generated in that wallet. Therefore, never expose the mnemonic as it may compromise all of your accounts.

This was section was inspired by The Pledge from Patrick Collins.

Comments

1. For public or external methods and variables, use NatSpec comments.
   - `forge doc` will parse these to autogenerate documentation.
   - Etherscan will display them in the contract UI.
2. For simple NatSpec comments, consider just documenting params in the docstring, such as `/// @notice Returns the sum of `x` and `y`.`, instead of using `@param` tags.
3. For complex NatSpec comments, consider using a tool like PlantUML to generate ASCII art diagrams to help explain complex aspects of the codebase.
4. Any markdown in your comments will carry over properly when generating docs with `forge doc`, so structure comments with markdown when useful.
   - Good: `/// @notice Returns the sum of `x` and `y`.`
   - Bad: `/// @notice Returns the sum of x and y.`

Resources

Write more secure code and better tests:

- transmissions11/solcurity
- nascentxyz/simple-security-toolkit

Foundry in Action:

- awesome-foundry: A curated list of awesome of the Foundry development framework.
- Nomad Monorepo: All the `contracts-*` packages. Good example of using many Foundry features including fuzzing, `ffi` and various cheatcodes.
- Sablier V2 Core: Another good example of many Foundry features. Also a pioneer of the state tree testing approach, see the `*.tree` files.
- Uniswap Periphery: Good example of using inheritance to isolate test fixtures.
- PRBMath: A library for fixed-point arithmetic in Solidity, with many parameterized tests that harness Foundry.

Creating an NFT with Solmate

This tutorial will walk you through creating an OpenSea compatible NFT with Foundry and Solmate. A full implementation of this tutorial can be found here.

This tutorial is for illustrative purposes only and provided on an as-is basis. The tutorial is not audited nor fully tested. No code in this tutorial should be used in a production environment.

Create project and install dependencies

Start by setting up a Foundry project following the steps outlined in the Getting started section. We will also install Solmate for their ERC721 implementation, as well as some OpenZeppelin utility libraries. Install the dependencies by running the following commands from the root of your project:

```
forge install transmissions11/solmate Openzeppelin/openzeppelin-contracts
```

These dependencies will be added as git submodules to your project.

If you have followed the instructions correctly your project should be structured like this:

```
→  hello-foundry git:(master) tree -L 2
.
├── foundry.toml
├── lib
│   ├── forge-std
│   ├── openzeppelin-contracts
│   └── solmate
├── script
│   └── Contract.s.sol
├── src
│   └── Contract.sol
└── test
    └── Contract.t.sol

7 directories, 4 files
```

Implement a basic NFT

We are then going to rename the boilerplate contract in `src/Contract.sol` to `src/NFT.sol` and replace the code:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.10;

import "solmate/tokens/ERC721.sol";
import "openzeppelin-contracts/contracts/utils/Strings.sol";

contract NFT is ERC721 {
    uint256 public currentTokenId;

    constructor(
        string memory _name,
        string memory _symbol
    ) ERC721(_name, _symbol) {}

    function mintTo(address recipient) public payable returns (uint256) {
        uint256 newItemId = ++currentTokenId;
        _safeMint(recipient, newItemId);
        return newItemId;
    }

    function tokenURI(uint256 id) public view virtual override returns (string
memory) {
        return Strings.toString(id);
    }
}
```

Let's take a look at this very basic implementation of an NFT. We start by importing two contracts from our git submodules. We import solmate's gas optimized implementation of the ERC721 standard which our NFT contract will inherit from. Our constructor takes the `_name` and `_symbol` arguments for our NFT and passes them on to the constructor of the parent ERC721 implementation. Lastly we implement the `mintTo` function which allows anyone to mint an NFT. This function increments the `currentTokenId` and makes use of the `_safeMint` function of our parent contract.

Compile & deploy with forge

To compile the NFT contract run `forge build`. You might experience a build failure due to wrong mapping:

```
Error:
Compiler run failed
error[6275]: ParserError: Source "lib/openzeppelin-
contracts/contracts/contracts/utils/Strings.sol" not found: File not found.
Searched the following locations: "/PATH/TO/REPO".
 --> src/NFT.sol:5:1:
  |
5 | import "openzeppelin-contracts/contracts/utils/Strings.sol";
  | ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

this can be fixed by setting up the correct remapping. Create a file `remappings.txt` in your project and add the line

```
openzeppelin-contracts/=lib/openzeppelin-contracts/
```

(You can find out more on remappings in the dependencies documentation.

By default the compiler output will be in the `out` directory. To deploy our compiled contract with Forge we have to set environment variables for the RPC endpoint and the private key we want to use to deploy.

Set your environment variables by running:

```
export RPC_URL=<Your RPC endpoint>
export PRIVATE_KEY=<Your wallets private key>
```

Once set, you can deploy your NFT with Forge by running the below command while adding the relevant constructor arguments to the NFT contract:

```
forge create NFT --rpc-url=$RPC_URL --private-key=$PRIVATE_KEY --constructor-
args <name> <symbol>
```

If successfully deployed, you will see the deploying wallet's address, the contract's address as well as the transaction hash printed to your terminal.

Minting NFTs from your contract

Calling functions on your NFT contract is made simple with Cast, Foundry's command-line tool for interacting with smart contracts, sending transactions, and getting chain data. Let's have a look at how we can use it to mint NFTs from our NFT contract.

Given that you already set your RPC and private key env variables during deployment, mint an NFT from your contract by running:

```
cast send --rpc-url=$RPC_URL <contractAddress>  "mintTo(address)" <arg> --
private-key=$PRIVATE_KEY
```

Well done! You just minted your first NFT from your contract. You can sanity check the owner of the NFT with `currentTokenId` equal to **1** by running the below `cast call` command. The address you provided above should be returned as the owner.

```
cast call --rpc-url=$RPC_URL --private-key=$PRIVATE_KEY <contractAddress>
"ownerOf(uint256)" 1
```

Extending our NFT contract functionality and testing

Let's extend our NFT by adding metadata to represent the content of our NFTs, as well as set a minting price, a maximum supply and the possibility to withdraw the collected proceeds from minting. To follow along you can replace your current NFT contract with the code snippet below:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity >=0.8.10;

import "solmate/tokens/ERC721.sol";
import "openzeppelin-contracts/contracts/utils/Strings.sol";
import "openzeppelin-contracts/contracts/access/Ownable.sol";

error MintPriceNotPaid();
error MaxSupply();
error NonExistentTokenURI();
error WithdrawTransfer();

contract NFT is ERC721, Ownable {

    using Strings for uint256;
    string public baseURI;
    uint256 public currentTokenId;
    uint256 public constant TOTAL_SUPPLY = 10_000;
    uint256 public constant MINT_PRICE = 0.08 ether;

    constructor(
        string memory _name,
        string memory _symbol,
        string memory _baseURI
    ) ERC721(_name, _symbol) {
        baseURI = _baseURI;
    }

    function mintTo(address recipient) public payable returns (uint256) {
        if (msg.value != MINT_PRICE) {
            revert MintPriceNotPaid();
        }
        uint256 newTokenId = currentTokenId + 1;
        if (newTokenId > TOTAL_SUPPLY) {
            revert MaxSupply();
        }
        currentTokenId = newTokenId;
        _safeMint(recipient, newTokenId);
        return newTokenId;
    }

    function tokenURI(uint256 tokenId)
        public
        view
        virtual
        override
        returns (string memory)
    {
        if (ownerOf(tokenId) == address(0)) {
            revert NonExistentTokenURI();
        }
        return
            bytes(baseURI).length > 0
                ? string(abi.encodePacked(baseURI, tokenId.toString()))
                : "";
    }
```

```
    function withdrawPayments(address payable payee) external onlyOwner {
        if (address(this).balance == 0) {
            revert WithdrawTransfer();
        }

        payable(payee).transfer(address(this).balance);
    }

    function _checkOwner() internal view override {
        require(msg.sender == owner(), "Ownable: caller is not the owner");
    }
}
```

Among other things, we have added metadata that can be queried from any front-end application like OpenSea, by calling the `tokenURI` method on our NFT contract.

> **Note**: If you want to provide a real URL to the constructor at deployment, and host the metadata of this NFT contract please follow the steps outlined here.

Let's test some of this added functionality to make sure it works as intended. Foundry offers an extremely fast EVM native testing framework through Forge.

Within your test folder rename the current `Contract.t.sol` test file to `NFT.t.sol`. This file will contain all tests regarding the NFT's `mintTo` method. Next, replace the existing boilerplate code with the below:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.10;

import "forge-std/Test.sol";
import "../src/NFT.sol";

contract NFTTest is Test {
    using stdStorage for StdStorage;

    NFT private nft;

    function setUp() public {
        // Deploy NFT contract
        nft = new NFT("NFT_tutorial", "TUT", "baseUri");
    }

    function test_RevertMintWithoutValue() public {
        vm.expectRevert(MintPriceNotPaid.selector);
        nft.mintTo(address(1));
    }

    function test_MintPricePaid() public {
        nft.mintTo{value: 0.08 ether}(address(1));
    }

    function test_RevertMintMaxSupplyReached() public {
        uint256 slot = stdstore
            .target(address(nft))
            .sig("currentTokenId()")
            .find();
        bytes32 loc = bytes32(slot);
        bytes32 mockedCurrentTokenId = bytes32(abi.encode(10000));
        vm.store(address(nft), loc, mockedCurrentTokenId);
        vm.expectRevert(MaxSupply.selector);
        nft.mintTo{value: 0.08 ether}(address(1));
    }

    function test_RevertMintToZeroAddress() public {
        vm.expectRevert("INVALID_RECIPIENT");
        nft.mintTo{value: 0.08 ether}(address(0));
    }

    function test_NewMintOwnerRegistered() public {
        nft.mintTo{value: 0.08 ether}(address(1));
        uint256 slotOfNewOwner = stdstore
            .target(address(nft))
            .sig(nft.ownerOf.selector)
            .with_key(address(1))
            .find();

        uint160 ownerOfTokenIdOne = uint160(
            uint256(
                (vm.load(address(nft), bytes32(abi.encode(slotOfNewOwner))))
            )
        );
        assertEq(address(ownerOfTokenIdOne), address(1));
    }
```

```
    function test_BalanceIncremented() public {
        nft.mintTo{value: 0.08 ether}(address(1));
        uint256 slotBalance = stdstore
            .target(address(nft))
            .sig(nft.balanceOf.selector)
            .with_key(address(1))
            .find();

        uint256 balanceFirstMint = uint256(
            vm.load(address(nft), bytes32(slotBalance))
        );
        assertEq(balanceFirstMint, 1);

        nft.mintTo{value: 0.08 ether}(address(1));
        uint256 balanceSecondMint = uint256(
            vm.load(address(nft), bytes32(slotBalance))
        );
        assertEq(balanceSecondMint, 2);
    }

    function test_SafeContractReceiver() public {
        Receiver receiver = new Receiver();
        nft.mintTo{value: 0.08 ether}(address(receiver));
        uint256 slotBalance = stdstore
            .target(address(nft))
            .sig(nft.balanceOf.selector)
            .with_key(address(receiver))
            .find();

        uint256 balance = uint256(vm.load(address(nft), bytes32(slotBalance)));
        assertEq(balance, 1);
    }

    function test_RevertUnSafeContractReceiver() public {
        // Adress set to 11, because first 10 addresses are restricted for
precompiles
        vm.etch(address(11), bytes("mock code"));
        vm.expectRevert(bytes(""));
        nft.mintTo{value: 0.08 ether}(address(11));
    }

    function test_WithdrawalWorksAsOwner() public {
        // Mint an NFT, sending eth to the contract
        Receiver receiver = new Receiver();
        address payable payee = payable(address(0x1337));
        uint256 priorPayeeBalance = payee.balance;
        nft.mintTo{value: nft.MINT_PRICE()}(address(receiver));
        // Check that the balance of the contract is correct
        assertEq(address(nft).balance, nft.MINT_PRICE());
        uint256 nftBalance = address(nft).balance;
        // Withdraw the balance and assert it was transferred
        nft.withdrawPayments(payee);
        assertEq(payee.balance, priorPayeeBalance + nftBalance);
    }

    function test_WithdrawalFailsAsNotOwner() public {
        // Mint an NFT, sending eth to the contract
```

```
        Receiver receiver = new Receiver();
        nft.mintTo{value: nft.MINT_PRICE()}(address(receiver));
        // Check that the balance of the contract is correct
        assertEq(address(nft).balance, nft.MINT_PRICE());
        // Confirm that a non-owner cannot withdraw
        vm.expectRevert("Ownable: caller is not the owner");
        vm.startPrank(address(0xd3ad));
        nft.withdrawPayments(payable(address(0xd3ad)));
        vm.stopPrank();
    }
}

contract Receiver is ERC721TokenReceiver {
    function onERC721Received(
        address operator,
        address from,
        uint256 id,
        bytes calldata data
    ) external override returns (bytes4) {
        return this.onERC721Received.selector;
    }
}
```

The test suite is set up as a contract with a `setUp` method which runs before every individual test.

As you can see, Forge offers a number of cheatcodes to manipulate state to accommodate your testing scenario.

For example, our `testFailMaxSupplyReached` test checks that an attempt to mint fails when the max supply of NFT is reached. Thus, the `currentTokenId` of the NFT contract needs to be set to the max supply by using the store cheatcode which allows you to write data to your contracts storage slots. The storage slots you wish to write to can easily be found using the `forge-std` helper library. You can run the test with the following command:

```
forge test
```

If you want to put your Forge skills to practice, write tests for the remaining methods of our NFT contract. Feel free to PR them to nft-tutorial, where you will find the full implementation of this tutorial.

Gas reports for your function calls

Foundry provides comprehensive gas reports about your contracts. For every function called within your tests, it returns the minimum, average, median and max gas cost. To print the gas report simply run:

```
forge test --gas-report
```

This comes in handy when looking at various gas optimizations within your contracts.

Let's have a look at the gas savings we made by substituting OpenZeppelin with Solmate for our ERC721 implementation. You can find the NFT implementation using both libraries here. Below are the resulting gas reports when running `forge test --gas-report` on that repository.

As you can see, our implementation using Solmate saves around 500 gas on a successful mint (the max gas cost of the `mintTo` function calls).

NftSolmate contract					
Deployment Cost	Deployment Size				
1583919	8489				
Function Name	min	avg	median	max	# calls
balanceOf	635	635	635	635	2
currentTokenId	2308	2308	2308	2308	1
mintTo	473	44574	69399	72745	9
ownerOf	566	566	566	566	1

NftOZ contract					
Deployment Cost	Deployment Size				
1791936	9528				
Function Name	min	avg	median	max	# calls
balanceOf	705	705	705	705	2
currentTokenId	2308	2308	2308	2308	1
mintTo	473	44797	69632	73214	9
ownerOf	602	602	602	602	1

That's it, I hope this will give you a good practical basis of how to get started with foundry. We think there is no better way to deeply understand solidity than writing your tests in solidity. You will also experience less context switching between javascript and solidity. Happy coding!

> Note: Follow this tutorial to learn how to deploy the NFT contract used here with solidity scripting.

Dockerizing a Foundry project

This tutorial shows you how to build, test, and deploy a smart contract using Foundry's Docker image. It adapts code from the solmate nft tutorial. If you haven't completed that tutorial yet, and are new to solidity, you may want to start with it first. Alternatively, if you have some familiarity with Docker and Solidity, you can use your own existing project and adjust accordingly. The full source code for both the NFT and the Docker stuff is available here.

> This tutorial is for illustrative purposes only and provided on an as-is basis. The tutorial is not audited nor fully tested. No code in this tutorial should be used in a production environment.

Installation and Setup

The only installation required to run this tutorial is Docker, and optionally, an IDE of your choice. Follow the Docker installation instructions.

To keep future commands succinct, let's re-tag the image:

```
docker tag ghcr.io/foundry-rs/foundry:latest foundry:latest
```

Having Foundry installed locally is not strictly required, but it may be helpful for debugging. You can install it using foundryup.

Finally, to use any of the `cast` or `forge create` portions of this tutorial, you will need access to an Ethereum node. If you don't have your own node running (likely), you can use a 3rd party node service. We won't recommend a specific provider in this tutorial. A good place to start learning about Nodes-as-a-Service is Ethereum's article on the subject.

For the rest of this tutorial, it is assumed that the RPC endpoint of your ethereum node is set like this: `export RPC_URL=<YOUR_RPC_URL>`

A tour around the Foundry docker image

The docker image can be used in two primary ways:

1. As an interface directly to forge and cast
2. As a base image for building your own containerized test, build, and deployment tooling

We will cover both, but let's start by taking a look at interfacing with foundry using docker. This is also a good test that your local installation worked!

We can run any of the `cast` commands against our docker image. Let's fetch the latest block information:

```
$ docker run foundry "cast block --rpc-url $RPC_URL latest"
baseFeePerGas        "0xb634241e3"
difficulty           "0x2e482bdf51572b"
extraData            "0x486976656f6e20686b"
gasLimit             "0x1c9c380"
gasUsed              "0x652993"
hash
"0x181748772da2f968bcc91940c8523bb6218a7d57669ded06648c9a9fb6839db5"
logsBloom
"0x40601004610000119 8c220108002b6064000294448140082108 20c0401280413184715008031
2500300051044208430002008029880029011520380060262400001c538d00440a885a02219d496
24aa110000003094500022c003600a00258009610c410323580032000849a0408a81a0a06010002
2505202280c61880c80020e08024440044040 4520d210429a0000400010089410c8408162903609
c920014028a94019088681018c909980701019201808040004100000080540610a9144d05002022
0c10a24c01c000002005400400022420140e18100400e10254926144c43a200cc00814208085408
810012884400301002 0c344402386a8c011819408"
miner                "0x1ad91ee08f21be3de0ba2ba6918e714da6b45836"
mixHash
"0xb920857687476c1bcb21557c5f6196762a46038924c5f82dc66300347a1cfc01"
nonce                "0x1ce6929033fbba90"
number               "0xdd3309"
parentHash
"0x39c6e1aa997d18a655c6317131589fd327ae814ef84e784f5eb1ab54b9941212"
receiptsRoot
"0x4724f3b270dcc970f141e493d8dc46aeba6fffe57688210051580ac960fe0037"
sealFields           []
sha3Uncles
"0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347"
size                 "0x1d6bb"
stateRoot
"0x0d4b714990132cf0f21801e2931b78454b26aad706fc6dc16b64e04f0c14737a"
timestamp            "0x6246259b"
totalDifficulty      "0x9923da68627095fd2e7"
transactions         [...]
uncles               []
```

If we're in a directory with some Solidity source code, we can mount that directory into docker and use `forge` however we wish. For example:

```
$ docker run -v $PWD:/app foundry "forge test --root /app --watch"
No files changed, compilation skipped

Ran 8 tests for test/OpenZeppelinNft.t.sol:OpenZeppelinNftTests
[PASS] testBalanceIncremented() (gas: 217829)
[PASS] testFailMaxSupplyReached() (gas: 134524)
[PASS] testFailMintToZeroAddress() (gas: 34577)
[PASS] testFailNoMintPricePaid() (gas: 5568)
[PASS] testFailUnSafeContractReceiver() (gas: 3524)
[PASS] testMintPricePaid() (gas: 81554)
[PASS] testNewMintOwnerRegistered() (gas: 190956)
[PASS] testSafeContractReceiver() (gas: 273132)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 1.39ms (1.47ms CPU
time)

Ran 8 tests for test/SolmateNft.sol:SolmateNftTests
[PASS] testBalanceIncremented() (gas: 217400)
[PASS] testFailMaxSupplyReached() (gas: 134524)
[PASS] testFailMintToZeroAddress() (gas: 34521)
[PASS] testFailNoMintPricePaid() (gas: 5568)
[PASS] testFailUnSafeContractReceiver() (gas: 3524)
[PASS] testMintPricePaid() (gas: 81321)
[PASS] testNewMintOwnerRegistered() (gas: 190741)
[PASS] testSafeContractReceiver() (gas: 272636)
Suite result: ok. 8 passed; 0 failed; 0 skipped; finished in 1.39ms (1.45ms CPU
time)

Ran 2 test suites in 3.69ms (2.78ms CPU time): 16 tests passed, 0 failed, 0
skipped (16 total tests)
```

You can see our code was compiled and tested entirely within the container. Also, since we passed the `--watch` option, the container will recompile the code whenever a change is detected.

> Note: The Foundry docker image is built on alpine and designed to be as slim as possible. For this reason, it does not currently include development resources like `git`. If you are planning to manage your entire development lifecycle within the container, you should build a custom development image on top of Foundry's image.

Creating a "build and test" image

Let's use the Foundry docker image as a base for using our own Docker image. We'll use the image to:

1. Build our solidity code
2. Run our solidity tests

A simple `Dockerfile` can accomplish these two goals:

```
# Use the latest foundry image
FROM ghcr.io/foundry-rs/foundry

# Copy our source code into the container
WORKDIR /app

# Build and test the source code
COPY . .
RUN forge build
RUN forge test
```

You can build this docker image and watch forge build/run the tests within the container:

```
$ docker build --no-cache --progress=plain .
```

Now, what happens if one of our tests fails? Modify `src/test/NFT.t.sol` as you please to make one of the tests fails. Try to build image again.

```
$ docker build --no-cache --progress=plain .
<...>
#9 0.522 Failed tests:
#9 0.522 [FAIL. Reason: Ownable: caller is not the owner]
testWithdrawalFailsAsNotOwner() (gas: 193917)
#9 0.522
#9 0.522 Encountered a total of 1 failing tests, 9 tests succeeded
------
error: failed to solve: executor failed running [/bin/sh -c forge test]: exit
code: 1
```

Our image failed to build because our tests failed! This is actually a nice property, because it means if we have a Docker image that successfully built (and therefore is available for use), we know the code inside the image passed the tests.*

---

*Of course, chain of custody of your docker images is very important. Docker layer hashes can be very useful for verification. In a production environment, consider signing your docker images.

Creating a deployer image

Now, we'll move on to a bit more of an advanced Dockerfile. Let's add an entrypoint that allows us to deploy our code by using the built (and tested!) image. We can target the Rinkeby testnet first.

```
# Use the latest foundry image
FROM ghcr.io/foundry-rs/foundry

# Copy our source code into the container
WORKDIR /app

# Build and test the source code
COPY . .
RUN forge build
RUN forge test

# Set the entrypoint to the forge deployment command
ENTRYPOINT ["forge", "create"]
```

Let's build the image, this time giving it a name:

```
$ docker build --no-cache --progress=plain -t nft-deployer .
```

Here's how we can use our docker image to deploy:

```
$ docker run nft-deployer --rpc-url $RPC_URL --constructor-args "ForgeNFT"
"FNFT" "https://ethereum.org" --private-key $PRIVATE_KEY ./src/NFT.sol:NFT
No files changed, compilation skipped
Deployer: 0x496e09fcb240c33b8fda3b4b74d81697c03b6b3d
Deployed to: 0x23d465eaa80ad2e5cdb1a2345e4b54edd12560d3
Transaction hash:
0xf88c68c4a03a86b0e7ecb05cae8dea36f2896cd342a6af978cab11101c6224a9
```

We've just built, tested, and deployed our contract entirely within a docker container! This tutorial was intended for testnet, but you can run the exact same Docker image targeting mainnet and be confident that the same code is being deployed by the same tooling.

Why is this useful?

Docker is about portability, reproducibility, and environment invariance. This means you can be less concerned about unexpected changes when you switch between environments, networks, developers, etc. Here are a few basic examples of why I like to use Docker images for smart contract deployment:

- Reduces overhead of ensuring system level dependencies match between deployment environments (e.g. does your production runner always have the same version of `forge` as your dev runner?)
- Increases confidence that code has been tested prior to deployment and has not been altered (e.g. if, in the above image, your code re-compiles on deployment, that's a major red flag).
- Eases pain points around segregation of duties: people with your mainnet credentials do not need to ensure they have the latest compiler, codebase, etc. It's easy to ensure

that the docker deploy image someone ran in testnet is identical to the one targeting mainnet.
- At the risk of sounding web2, Docker is an accepted standard on virtually all public cloud providers. It makes it easy to schedule jobs, tasks, etc that need to interact with the blockchain.

Troubleshooting

As noted above, the Foundry image does not include `git` by default. This can cause certain commands to fail without a clear cause. For example:

```
$ docker run foundry "forge init --no-git /test"
Initializing /test...
Installing ds-test in "/test/lib/ds-test", (url: https://github.com/dapphub/ds-test, tag: None)
Error:
   0: No such file or directory (os error 2)

Location:
   cli/src/cmd/forge/install.rs:107
```

In this case, the failure is still caused by a missing `git` installation. The recommended fix is to build off the existing Foundry image and install any additional development dependencies you need.

Testing EIP-712 Signatures

Intro

EIP-712 introduced the ability to sign transactions off-chain which other users can later execute on-chain. A common example is EIP-2612 gasless token approvals.

Traditionally, setting a user or contract allowance to transfer ERC-20 tokens from an owner's balance required the owner to submit an approval on-chain. As this proved to be poor UX, DAI introduced ERC-20 `permit` (later standardized as EIP-2612) allowing the owner to sign the approval *off-chain* which the spender (or anyone else!) can submit on-chain prior to the `transferFrom`.

This guide will cover testing this pattern in Solidity using Foundry.

Diving In

First we'll cover a basic token transfer:

- Owner signs approval off-chain
- Spender calls `permit` and `transferFrom` on-chain

We'll use Solmate's ERC-20, as EIP-712 and EIP-2612 batteries come included. Take a glance over the full contract if you haven't already - here is `permit` implemented:

```
    /*///////////////////////////////////////////////////////////////
                              EIP-2612 LOGIC
    ///////////////////////////////////////////////////////////////*/

    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) public virtual {
        require(deadline >= block.timestamp, "PERMIT_DEADLINE_EXPIRED");

        // Unchecked because the only math done is incrementing
        // the owner's nonce which cannot realistically overflow.
        unchecked {
            address recoveredAddress = ecrecover(
                keccak256(
                    abi.encodePacked(
                        "\x19\x01",
                        DOMAIN_SEPARATOR(),
                        keccak256(
                            abi.encode(
                                keccak256(
                                    "Permit(address owner,address
spender,uint256 value,uint256 nonce,uint256 deadline)"
                                ),
                                owner,
                                spender,
                                value,
                                nonces[owner]++,
                                deadline
                            )
                        )
                    )
                ),
                v,
                r,
                s
            );

            require(recoveredAddress != address(0) && recoveredAddress ==
owner, "INVALID_SIGNER");

            allowance[recoveredAddress][spender] = value;
        }

        emit Approval(owner, spender, value);
    }
```

We'll also be using a custom `SigUtils` contract to help create, hash, and sign the approvals off-chain.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.20;

contract SigUtils {
    bytes32 internal DOMAIN_SEPARATOR;

    constructor(bytes32 _DOMAIN_SEPARATOR) {
        DOMAIN_SEPARATOR = _DOMAIN_SEPARATOR;
    }

    // keccak256("Permit(address owner,address spender,uint256 value,uint256
nonce,uint256 deadline)");
    bytes32 public constant PERMIT_TYPEHASH =
        0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64845d6126c9;

    struct Permit {
        address owner;
        address spender;
        uint256 value;
        uint256 nonce;
        uint256 deadline;
    }

    // computes the hash of a permit
    function getStructHash(Permit memory _permit)
        internal
        pure
        returns (bytes32)
    {
        return
            keccak256(
                abi.encode(
                    PERMIT_TYPEHASH,
                    _permit.owner,
                    _permit.spender,
                    _permit.value,
                    _permit.nonce,
                    _permit.deadline
                )
            );
    }

    // computes the hash of the fully encoded EIP-712 message for the domain,
which can be used to recover the signer
    function getTypedDataHash(Permit memory _permit)
        public
        view
        returns (bytes32)
    {
        return
            keccak256(
                abi.encodePacked(
                    "\x19\x01",
                    DOMAIN_SEPARATOR,
                    getStructHash(_permit)
                )
            );
```

```
        }
    }
```

Handling Dynamic Values

While the Permit struct passed in the getStructHash() function above doesn't contain any dynamic value types, if you're using them it's important to remember that 'bytes' and 'string' types must be encoded as a 'keccak256' hash of their contents. More on this aspect of the EIP 712 Spec here.

Setup

- Deploy a mock ERC-20 token and `SigUtils` helper with the token's EIP-712 domain separator
- Create private keys to mock the owner and spender
- Derive their addresses using the `vm.addr` cheatcode
- Mint the owner a test token

```
contract Token_ERC20 is MockERC20 {
    constructor(string memory name, string memory symbol, uint8 decimals) {
        initialize(name, symbol, decimals);
    }

    function mint(address to, uint256 value) public virtual {
        _mint(to, value);
    }

    function burn(address from, uint256 value) public virtual {
        _burn(from, value);
    }
}

contract ERC20Test is Test {
    Token_ERC20 internal token;
    SigUtils internal sigUtils;

    uint256 internal ownerPrivateKey;
    uint256 internal spenderPrivateKey;

    address internal owner;
    address internal spender;

    function setUp() public {
        token = new Token_ERC20("Token", "TKN", 18);
        sigUtils = new SigUtils(token.DOMAIN_SEPARATOR());

        ownerPrivateKey = 0xA11CE;
        spenderPrivateKey = 0xB0B;

        owner = vm.addr(ownerPrivateKey);
        spender = vm.addr(spenderPrivateKey);

        token.mint(owner, 1e18);
    }
```

Testing: `permit`

- Create an approval for the spender
- Compute its digest using `sigUtils.getTypedDataHash`
- Sign the digest using the `vm.sign` cheatcode with the owner's private key
- Store the `uint8 v, bytes32 r, bytes32 s` of the signature
- Call `permit` with the signed permit and signature to execute the approval on-chain

```
function test_Permit() public {
    SigUtils.Permit memory permit = SigUtils.Permit({
        owner: owner,
        spender: spender,
        value: 1e18,
        nonce: 0,
        deadline: 1 days
    });

    bytes32 digest = sigUtils.getTypedDataHash(permit);

    (uint8 v, bytes32 r, bytes32 s) = vm.sign(ownerPrivateKey, digest);

    token.permit(
        permit.owner,
        permit.spender,
        permit.value,
        permit.deadline,
        v,
        r,
        s
    );

    assertEq(token.allowance(owner, spender), 1e18);
    assertEq(token.nonces(owner), 1);
}
```

- Ensure failure for calls with an expired deadline, invalid signer, invalid nonce, and signature replay

```
    function testRevert_ExpiredPermit() public {
        SigUtils.Permit memory permit = SigUtils.Permit({
            owner: owner,
            spender: spender,
            value: 1e18,
            nonce: token.nonces(owner),
            deadline: 1 days
        });

        bytes32 digest = sigUtils.getTypedDataHash(permit);

        (uint8 v, bytes32 r, bytes32 s) = vm.sign(ownerPrivateKey, digest);

        vm.warp(1 days + 1 seconds); // fast forward one second past the
deadline

        vm.expectRevert("PERMIT_DEADLINE_EXPIRED");
        token.permit(
            permit.owner,
            permit.spender,
            permit.value,
            permit.deadline,
            v,
            r,
            s
        );
    }

    function testRevert_InvalidSigner() public {
        SigUtils.Permit memory permit = SigUtils.Permit({
            owner: owner,
            spender: spender,
            value: 1e18,
            nonce: token.nonces(owner),
            deadline: 1 days
        });

        bytes32 digest = sigUtils.getTypedDataHash(permit);

        (uint8 v, bytes32 r, bytes32 s) = vm.sign(spenderPrivateKey, digest);
// spender signs owner's approval

        vm.expectRevert("INVALID_SIGNER");
        token.permit(
            permit.owner,
            permit.spender,
            permit.value,
            permit.deadline,
            v,
            r,
            s
        );
    }

    function testRevert_InvalidNonce() public {
        SigUtils.Permit memory permit = SigUtils.Permit({
            owner: owner,
```

```
            spender: spender,
            value: 1e18,
            nonce: 1, // owner nonce stored on-chain is 0
            deadline: 1 days
        });

        bytes32 digest = sigUtils.getTypedDataHash(permit);

        (uint8 v, bytes32 r, bytes32 s) = vm.sign(ownerPrivateKey, digest);

        vm.expectRevert("INVALID_SIGNER");
        token.permit(
            permit.owner,
            permit.spender,
            permit.value,
            permit.deadline,
            v,
            r,
            s
        );
    }

    function testRevert_SignatureReplay() public {
        SigUtils.Permit memory permit = SigUtils.Permit({
            owner: owner,
            spender: spender,
            value: 1e18,
            nonce: 0,
            deadline: 1 days
        });

        bytes32 digest = sigUtils.getTypedDataHash(permit);

        (uint8 v, bytes32 r, bytes32 s) = vm.sign(ownerPrivateKey, digest);

        token.permit(
            permit.owner,
            permit.spender,
            permit.value,
            permit.deadline,
            v,
            r,
            s
        );

        vm.expectRevert("INVALID_SIGNER");
        token.permit(
            permit.owner,
            permit.spender,
            permit.value,
            permit.deadline,
            v,
            r,
            s
        );
    }
```

Testing: `transferFrom`

- Create, sign, and execute an approval for the spender
- Call `tokenTransfer` as the spender using the `vm.prank` cheatcode to execute the transfer

```
    function test_TransferFromLimitedPermit() public {
        SigUtils.Permit memory permit = SigUtils.Permit({
            owner: owner,
            spender: spender,
            value: 1e18,
            nonce: 0,
            deadline: 1 days
        });

        bytes32 digest = sigUtils.getTypedDataHash(permit);

        (uint8 v, bytes32 r, bytes32 s) = vm.sign(ownerPrivateKey, digest);

        token.permit(
            permit.owner,
            permit.spender,
            permit.value,
            permit.deadline,
            v,
            r,
            s
        );

        vm.prank(spender);
        token.transferFrom(owner, spender, 1e18);

        assertEq(token.balanceOf(owner), 0);
        assertEq(token.balanceOf(spender), 1e18);
        assertEq(token.allowance(owner, spender), 0);
    }

    function test_TransferFromMaxPermit() public {
        SigUtils.Permit memory permit = SigUtils.Permit({
            owner: owner,
            spender: spender,
            value: type(uint256).max,
            nonce: 0,
            deadline: 1 days
        });

        bytes32 digest = sigUtils.getTypedDataHash(permit);

        (uint8 v, bytes32 r, bytes32 s) = vm.sign(ownerPrivateKey, digest);

        token.permit(
            permit.owner,
            permit.spender,
            permit.value,
            permit.deadline,
            v,
            r,
            s
        );

        vm.prank(spender);
        token.transferFrom(owner, spender, 1e18);
```

```
        assertEq(token.balanceOf(owner), 0);
        assertEq(token.balanceOf(spender), 1e18);
        assertEq(token.allowance(owner, spender), type(uint256).max);
    }
```

- Ensure failure for calls with an invalid allowance and invalid balance

```
    function testFail_InvalidAllowance() public {
        SigUtils.Permit memory permit = SigUtils.Permit({
            owner: owner,
            spender: spender,
            value: 5e17, // approve only 0.5 tokens
            nonce: 0,
            deadline: 1 days
        });

        bytes32 digest = sigUtils.getTypedDataHash(permit);

        (uint8 v, bytes32 r, bytes32 s) = vm.sign(ownerPrivateKey, digest);

        token.permit(
            permit.owner,
            permit.spender,
            permit.value,
            permit.deadline,
            v,
            r,
            s
        );

        vm.prank(spender);
        token.transferFrom(owner, spender, 1e18); // attempt to transfer 1
token
    }

    function testFail_InvalidBalance() public {
        SigUtils.Permit memory permit = SigUtils.Permit({
            owner: owner,
            spender: spender,
            value: 2e18, // approve 2 tokens
            nonce: 0,
            deadline: 1 days
        });

        bytes32 digest = sigUtils.getTypedDataHash(permit);

        (uint8 v, bytes32 r, bytes32 s) = vm.sign(ownerPrivateKey, digest);

        token.permit(
            permit.owner,
            permit.spender,
            permit.value,
            permit.deadline,
            v,
            r,
            s
        );

        vm.prank(spender);
        token.transferFrom(owner, spender, 2e18); // attempt to transfer 2
tokens (owner only owns 1)
    }
```

Bundled Example

Here is a section of a mock contract that just deposits ERC-20 tokens. Note how `deposit` requires a preliminary `approve` or `permit` tx in order to transfer tokens, while `depositWithPermit` sets the allowance *and* transfers the tokens in a single tx.

```
    ///                                                          ///
    ///                          DEPOSIT                         ///
    ///                                                          ///

    /// @notice Deposits ERC-20 tokens (requires pre-approval)
    /// @param _tokenContract The ERC-20 token address
    /// @param _amount The number of tokens
    function deposit(address _tokenContract, uint256 _amount) external {
        ERC20(_tokenContract).transferFrom(msg.sender, address(this), _amount);

        userDeposits[msg.sender][_tokenContract] += _amount;

        emit TokenDeposit(msg.sender, _tokenContract, _amount);
    }

    ///                                                          ///
    ///                      DEPOSIT w/ PERMIT                   ///
    ///                                                          ///

    /// @notice Deposits ERC-20 tokens with a signed approval
    /// @param _tokenContract The ERC-20 token address
    /// @param _amount The number of tokens to transfer
    /// @param _owner The user signing the approval
    /// @param _spender The user to transfer the tokens (ie this contract)
    /// @param _value The number of tokens to approve the spender
    /// @param _deadline The timestamp the permit expires
    /// @param _v The 129th byte and chain id of the signature
    /// @param _r The first 64 bytes of the signature
    /// @param _s Bytes 64-128 of the signature
    function depositWithPermit(
        address _tokenContract,
        uint256 _amount,
        address _owner,
        address _spender,
        uint256 _value,
        uint256 _deadline,
        uint8 _v,
        bytes32 _r,
        bytes32 _s
    ) external {
        ERC20(_tokenContract).permit(
            _owner,
            _spender,
            _value,
            _deadline,
            _v,
            _r,
            _s
        );

        ERC20(_tokenContract).transferFrom(_owner, address(this), _amount);

        userDeposits[_owner][_tokenContract] += _amount;

        emit TokenDeposit(_owner, _tokenContract, _amount);
    }
```

Setup

- Deploy the `Deposit` contract, a mock ERC-20 token, and `SigUtils` helper with the token's EIP-712 domain separator
- Create a private key to mock the owner (the spender is now the `Deposit` address)
- Derive the owner address using the `vm.addr` cheatcode
- Mint the owner a test token

```
contract DepositTest is Test {
    Deposit internal deposit;
    MockERC20 internal token;
    SigUtils internal sigUtils;

    uint256 internal ownerPrivateKey;
    address internal owner;

    function setUp() public {
        deposit = new Deposit();
        token = new MockERC20();
        sigUtils = new SigUtils(token.DOMAIN_SEPARATOR());

        ownerPrivateKey = 0xA11CE;
        owner = vm.addr(ownerPrivateKey);

        token.mint(owner, 1e18);
    }
```

Testing: `depositWithPermit`

- Create an approval for the `Deposit` contract
- Compute its digest using `sigUtils.getTypedDataHash`
- Sign the digest using the `vm.sign` cheatcode with the owner's private key
- Store the `uint8 v, bytes32 r, bytes32 s` of the signature
  - *Note:* can convert to bytes via `bytes signature = abi.encodePacked(r, s, v)`
- Call `depositWithPermit` with the signed approval and signature to transfer the tokens into the contract

```
    function test_DepositWithLimitedPermit() public {
        SigUtils.Permit memory permit = SigUtils.Permit({
            owner: owner,
            spender: address(deposit),
            value: 1e18,
            nonce: token.nonces(owner),
            deadline: 1 days
        });

        bytes32 digest = sigUtils.getTypedDataHash(permit);

        (uint8 v, bytes32 r, bytes32 s) = vm.sign(ownerPrivateKey, digest);

        deposit.depositWithPermit(
            address(token),
            1e18,
            permit.owner,
            permit.spender,
            permit.value,
            permit.deadline,
            v,
            r,
            s
        );

        assertEq(token.balanceOf(owner), 0);
        assertEq(token.balanceOf(address(deposit)), 1e18);

        assertEq(token.allowance(owner, address(deposit)), 0);
        assertEq(token.nonces(owner), 1);

        assertEq(deposit.userDeposits(owner, address(token)), 1e18);
    }

    function test_DepositWithMaxPermit() public {
        SigUtils.Permit memory permit = SigUtils.Permit({
            owner: owner,
            spender: address(deposit),
            value: type(uint256).max,
            nonce: token.nonces(owner),
            deadline: 1 days
        });

        bytes32 digest = sigUtils.getTypedDataHash(permit);

        (uint8 v, bytes32 r, bytes32 s) = vm.sign(ownerPrivateKey, digest);

        deposit.depositWithPermit(
            address(token),
            1e18,
            permit.owner,
            permit.spender,
            permit.value,
            permit.deadline,
            v,
            r,
            s
```

```
        );

        assertEq(token.balanceOf(owner), 0);
        assertEq(token.balanceOf(address(deposit)), 1e18);

        assertEq(token.allowance(owner, address(deposit)), type(uint256).max);
        assertEq(token.nonces(owner), 1);

        assertEq(deposit.userDeposits(owner, address(token)), 1e18);
    }
```

- Ensure failure for invalid `permit` and `transferFrom` calls as previously shown

TLDR

Use Foundry cheatcodes `addr`, `sign`, and `prank` to test EIP-712 signatures in Foundry.

All source code can be found here.

Solidity Scripting

Introduction

Solidity scripting is a way to declaratively deploy contracts using Solidity, instead of using the more limiting and less user friendly `forge create`.

Solidity scripts are like the scripts you write when working with tools like Hardhat; what makes Solidity scripting different is that they are written in Solidity instead of JavaScript, and they are run on the fast Foundry EVM backend, which provides dry-run capabilities.

High Level Overview

`forge script` does not work in a sync manner. First, it collects all transactions from the script, and only then does it broadcast them all. It can essentially be split into 4 phases:

1. Local Simulation - The contract script is run in a local evm. If a rpc/fork url has been provided, it will execute the script in that context. Any **external call** (not static, not internal) from a `vm.broadcast` and/or `vm.startBroadcast` will be appended to a list.
2. Onchain Simulation - Optional. If a rpc/fork url has been provided, then it will sequentially execute all the collected transactions from the previous phase here.
3. Broadcasting - Optional. If the `--broadcast` flag is provided and the previous phases have succeeded, it will broadcast the transactions collected at step `1`. and simulated at step `2`.
4. Verification - Optional. If the `--verify` flag is provided, there's an API key, and the previous phases have succeeded it will attempt to verify the contract. (eg. etherscan).

Given this flow, it's important to be aware that transactions whose behaviour can be influenced by external state/actors might have a different result than what was simulated on step `2`. Eg. frontrunning.

Set Up

Let's try to deploy the NFT contract made in the solmate tutorial with solidity scripting. First of all, we would need to create a new Foundry project via:

```
forge init solidity-scripting
```

Since the NFT contract from the solmate tutorial inherits both `solmate` and `OpenZeppelin` contracts, we'll have to install them as dependencies by running:

```
# Enter the project
cd solidity-scripting

# Install Solmate and OpenZeppelin contracts as dependencies
forge install transmissions11/solmate Openzeppelin/openzeppelin-contracts@v5.0.1
```

Next, we have to delete the `Counter.sol` file in the `src` folder and create another file called `NFT.sol`. You can do this by running:

```
rm src/Counter.sol test/Counter.t.sol && touch src/NFT.sol && ls src
```

Once that's done, you should open up your preferred code editor and copy the code below into the `NFT.sol` file.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity >=0.8.10;

import "solmate/tokens/ERC721.sol";
import "openzeppelin-contracts/contracts/utils/Strings.sol";
import "openzeppelin-contracts/contracts/access/Ownable.sol";

error MintPriceNotPaid();
error MaxSupply();
error NonExistentTokenURI();
error WithdrawTransfer();

contract NFT is ERC721, Ownable {

    using Strings for uint256;
    string public baseURI;
    uint256 public currentTokenId;
    uint256 public constant TOTAL_SUPPLY = 10_000;
    uint256 public constant MINT_PRICE = 0.08 ether;

    constructor(
        string memory _name,
        string memory _symbol,
        string memory _baseURI
    ) ERC721(_name, _symbol) Ownable(msg.sender) {
        baseURI = _baseURI;
    }

    function mintTo(address recipient) public payable returns (uint256) {
        if (msg.value != MINT_PRICE) {
            revert MintPriceNotPaid();
        }
        uint256 newTokenId = ++currentTokenId;
        if (newTokenId > TOTAL_SUPPLY) {
            revert MaxSupply();
        }
        _safeMint(recipient, newTokenId);
        return newTokenId;
    }

    function tokenURI(uint256 tokenId)
        public
        view
        virtual
        override
        returns (string memory)
    {
        if (ownerOf(tokenId) == address(0)) {
            revert NonExistentTokenURI();
        }
        return
            bytes(baseURI).length > 0
                ? string(abi.encodePacked(baseURI, tokenId.toString()))
                : "";
    }

    function withdrawPayments(address payable payee) external onlyOwner {
```

```
        uint256 balance = address(this).balance;
        (bool transferTx, ) = payee.call{value: balance}("");
        if (!transferTx) {
            revert WithdrawTransfer();
        }
    }
}
```

Now, let's try compiling our contract to make sure everything is in order.

```
forge build
```

If your output looks like this, the contracts successfully compiled.

Deploying our contract

We're going to deploy the `NFT` contract to the Sepolia testnet, but to do this we'll need to configure Foundry a bit, by setting things like a Sepolia RPC URL, the private key of an account that's funded with Sepolia Eth, and an Etherscan key for the verification of the NFT contract.

💡 Note: You can get some Sepolia testnet ETH here .

Environment Configuration

Once you have all that create a `.env` file and add the variables. Foundry automatically loads in a `.env` file present in your project directory.

The .env file should follow this format:

```
SEPOLIA_RPC_URL=
PRIVATE_KEY=
ETHERSCAN_API_KEY=
```

We now need to edit the `foundry.toml` file. There should already be one in the root of the project.

Add the following lines to the end of the file:

```
[rpc_endpoints]
sepolia = "${SEPOLIA_RPC_URL}"

[etherscan]
sepolia = { key = "${ETHERSCAN_API_KEY}" }
```

This creates a RPC alias for Sepolia and loads the Etherscan API key.

Writing the Script

Next, we have to create a folder and name it `script` and create a file in it called `NFT.s.sol`. This is where we will create the deployment script itself.

The contents of `NFT.s.sol` should look like this:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;

import "forge-std/Script.sol";
import "../src/NFT.sol";

contract MyScript is Script {
    function run() external {
        uint256 deployerPrivateKey = vm.envUint("PRIVATE_KEY");
        vm.startBroadcast(deployerPrivateKey);

        NFT nft = new NFT("NFT_tutorial", "TUT", "baseUri");

        vm.stopBroadcast();
    }
}
```

Now let's read through the code and figure out what it actually means and does.

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.13;
```

Remember even if it's a script it still works like a smart contract, but is never deployed, so just like any other smart contract written in Solidity the `pragma version` has to be specified.

```
import "forge-std/Script.sol";
import "../src/NFT.sol";
```

Just like we may import Forge Std to get testing utilities when writing tests, Forge Std also provides some scripting utilities that we import here.

The next line just imports the `NFT` contract.

```
contract MyScript is Script {
```

We have created a contract called `MyScript` and it inherits `Script` from Forge Std.

```
function run() external {
```

By default, scripts are executed by calling the function named `run`, our entrypoint.

```
uint256 deployerPrivateKey = vm.envUint("PRIVATE_KEY");
```

This loads in the private key from our `.env` file. **Note:** you must be careful when exposing private keys in a `.env` file and loading them into programs. This is only recommended for use with non-privileged deployers or for local / test setups. For production setups please review the various wallet options that Foundry supports.

```
vm.startBroadcast(deployerPrivateKey);
```

This is a special cheatcode that records calls and contract creations made by our main script contract. We pass the `deployerPrivateKey` in order to instruct it to use that key for signing the transactions. Later, we will broadcast these transactions to deploy our NFT contract.

```
NFT nft = new NFT("NFT_tutorial", "TUT", "baseUri");
```

Here we have just created our NFT contract. Because we called `vm.startBroadcast()` before this line, the contract creation will be recorded by Forge, and as mentioned previously, we can broadcast the transaction to deploy the contract on-chain. The broadcast transaction logs will be stored in the `broadcast` directory by default. You can change the logs location by setting `broadcast` in your `foundry.toml` file.

Now that you're up to speed about what the script smart contract does, let's run it.

You should have added the variables we mentioned earlier to the `.env` for this next part to work.

At the root of the project run:

```
# To load the variables in the .env file
source .env

# To deploy and verify our contract
forge script script/NFT.s.sol:MyScript --rpc-url $SEPOLIA_RPC_URL --broadcast --verify -vvvv
```

Forge is going to run our script and broadcast the transactions for us - this can take a little while, since Forge will also wait for the transaction receipts. You should see something like this after a minute or so:

```
Paid: 0.00397569001192707 ETH (1325230 gas * 3.000000009 gwei)


Transactions saved to: broadcast/NFT.s.sol/4/run-latest.json



==========

ONCHAIN EXECUTION COMPLETE & SUCCESSFUL. Transaction receipts written to "broadcast/NFT.s.sol/4/run-latest.json"
##
Start Contract Verification

Submitting verification for [src/NFT.sol:NFT] 0x5537843691cfa63475553d68e85cf84b334541b5.
Submitted contract for verification:
        Response: `OK`
        GUID: `f5sbmpjmhknjsfdwjdcyhvwqst8wayzvmrmuhebmg7em8kwams`
        URL: https://rinkeby.etherscan.io/address/0x5537843691cfa63475553d68e85cf84b334541b5
Waiting for verification result...
Contract successfully verified.

Transactions saved to: broadcast/NFT.s.sol/4/run-latest.json

→  solidity-scripting git:(master) ✗
```

This confirms that you have successfully deployed the `NFT` contract to the Sepolia testnet and have also verified it on Etherscan, all with one command.

Deploying locally

You can deploy to Anvil, the local testnet, by configuring the port as the `fork-url`.

Here, we have two options in terms of accounts. We can either start anvil without any flags and use one of the private keys provided. Or, we can pass a mnemonic to anvil to use.

Using Anvil's Default Accounts

First, start Anvil:

```
anvil
```

Update your `.env` file with a private key given to you by Anvil.

Then run the following script:

```
forge script script/NFT.s.sol:MyScript --fork-url http://localhost:8545 --broadcast
```

Using a Custom Mnemonic

Add the following line to your `.env` file and complete it with your mnemonic:

```
MNEMONIC=
```

It is expected that the `PRIVATE_KEY` environment variable we set earlier is one of the first 10 accounts in this mnemonic.

Start Anvil with the custom mnemonic:

```
source .env

anvil -m $MNEMONIC
```

Then run the following script:

```
forge script script/NFT.s.sol:MyScript --fork-url http://localhost:8545 --broadcast
```

💡 Note: A full implementation of this tutorial can be found here and for further reading about solidity scripting, you can check out the `forge script` reference.

Deterministic deployment using CREATE2

Introduction

Enshrined into the EVM as part of the Constantinople fork of 2019, `CREATE2` is an opcode that started its journey as EIP-1014. `CREATE2` allows you to deploy smart contracts to deterministic addresses, based on parameters controlled by the deployer. As a result, it's often mentioned as enabling "counterfactual" deployments, where you can interact with an addresses that haven't been created yet because CREATE2 guarantees known code can be placed at that address. This is in contrast to the `CREATE` opcode, where the address of the deployed contract is a function of the deployer's nonce. With `CREATE2`, you can use the same deployer account to deploy contracts to the same address across multiple networks, even if the address has varying nonces.

In this tutorial, we will:

1. Look at a `CREATE2` factory implementation.
2. Deploy the factory using the traditional deployment methods.
3. Use this deployed factory to in turn deploy a simple counter contract at a deterministic address.
4. Simulate this set of events by writing a simple test in Foundry.

Prerequisites

1. Some familiarity with Solidity and Foundry is required, and some familiarity with the inline assembly is recommended. Refer to the official Solidity docs for a primer on inline assembly.
2. Make sure you have Foundry installed on your system.
3. Initialize a new Foundry project.

CREATE2 Factory

Create a file named `Create2.sol` Inside the `src` directory. Initialize a contract named `Create2` like this:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.20;

contract Create2 {

    error Create2EmptyBytecode();

    error Create2FailedDeployment();
}
```

These errors are meant to enforce some sanity checks on the factory deployment, and revert the whole transaction when triggered. The `Create2EmptyBytecode()` error triggers if the bytecode passed to the `deploy` function is empty, and the `Create2FailedDeployment()` error triggers if the deployment fails for any reason.

> ℹ️ **Note**
>
> Please note that a `CREATE2` deployment may fail due to a number of reasons. For example, if the bytecode is invalid, or if a contract is already deployed at the computed address. Your deployment may also fail if your constructor reverts for any reason.

Next, create a function named `deploy`:

```
function deploy(bytes32 salt, bytes memory creationCode) external payable
returns (address addr) {

 }
```

This function takes 2 inputs:

1. The `salt` used to calculate the final address. This can basically be any random value we want it to be.
2. The creation code of the contract that we want to deploy.

The address of the newly deployed contract is the returned after a successful deploy.

> ℹ️ **Note**
>
> You can send ETH to a contract that is being deployed using `CREATE2`, but only if it has a payable constructor. If you try to send ETH to it without a payable constructor, the transaction will revert.

Add this revert statement at the top of the deploy function. We want our function to reject the deploy request if the following conditions are not met:

```
    if (creationCode.length == 0) {
        revert Create2EmptyBytecode();
    }
```

Next, we will call the `CREATE2` opcode using inline assembly, which can be done using the `assembly` keyword:

To call the CREATE2 opcode from inline assembly, we need to pass in 4 parameters:

```
    assembly {
        addr := create2(callvalue(), add(creationCode, 0x20),
mload(creationCode), salt)
    }
```

1. The amount of ETH that we want to send to the new address as part of the deployment. Here we just pass in the `callvalue()`, which is the amount of ETH sent to the factory contract as part of the transaction. Think of it as a lower level version of `msg.value`.
2. The second and third parameters refer to the range of memory our bytecode is located in. `add(bytecode, 0x20)` takes in a reference to the location of the `bytes` variable bytecode in memory, and skips 32 bytes (0x20 in hex) to point to the actual bytecode.

Note

The `bytes` type in Solidity is a dynamically sized byte array, where the first 32 bytes of memory represent the length of the array, and the remaining bytes represent the actual data. Therefore, when we pass in a reference to `bytes` variable, we need to skip the first 32 bytes to point to the actual data. Read more about the `add` and `mload` opcodes at evm.codes.

Finally, we will revert the whole transaction if the deployment fails for any reason, in which case the `CREATE2` opcode will return a 0 address:

```
        if (addr == address(0)) {
            revert Create2FailedDeployment();
        }
```

Lastly, let us create a view function named `computeAddress`. This function should take in the `salt` and `creationCode` as parameters, and return the address of the contract that would be deployed using the `deploy` function:

```
function computeAddress(bytes32 salt, bytes32 creationCodeHash) external view
returns (address addr) {

 }
```

Inside the function, paste the following code that uses inline assembly to calculate the address by performing the same calculations as the `CREATE2` opcode would:

```
address contractAddress = address(this);

assembly {
    let ptr := mload(0x40)

    mstore(add(ptr, 0x40), creationCodeHash)
    mstore(add(ptr, 0x20), salt)
    mstore(ptr, contractAddress)
    let start := add(ptr, 0x0b)
    mstore8(start, 0xff)
    addr := keccak256(start, 85)
}
```

Before trying to understand the assembly code here, let us take a look at the formula that the `CREATE2` opcode uses to calculate the address:

```
keccak256(0xff ++ address ++ salt ++ keccak256(bytecode))[12:]
```

`0xff` is a hardcoded prefix that prevents hash-collision between addresses that are deployed using `CREATE` and `CREATE2`. The `address` param refers to the address of the contract that is calling the `CREATE2` opcode, in our case the factory contract. These 4 params are concatenated together, and `keccak256` is used to generate a 32 byte hash. The first 12 bytes are truncated, and the remaining 20 bytes are used as the address of the deployed contract.

The entirety of the assembly code in the `computeAddress` function is an attempt at recreating the same formula, albeit without calling the `CREATE2` opcode:

1. `mload(0x40)` loads the free memory pointer into memory. This is the pointer that points to the next free memory slot in the memory array. Read more about this in Solidity docs.
2. `mstore(add(ptr, 0x40), bytecodeHash)` stores the `bytecodeHash` starting at the memory location pointed to by `ptr + 0x40`, i.e. `ptr+ 64 bytes`.
3. `mstore(add(ptr, 0x20), salt)` stores the `salt` at the memory location pointed to by `ptr + 0x20`.
4. `mstore(ptr, contractAddress)` stores the `contractAddress` at the memory location pointed to by `ptr`.

ℹ️ Note

Recall that all the params passed to the `computeAddress` function are 32 bytes long, and are stored in memory as 32 byte values. However an address in Solidity is 20 bytes

long, and is stored in memory as a 32 byte value, where the first 12 bytes are replaced by 0s. Therefore, when we need to skip 12 bytes to point to the actual address.

---

5. `let start := add(ptr, 0x0b)` creates a new variable `start` that points to the memory location `ptr + 0x0b`, i.e. `ptr + 11 bytes`. 6.Lastly, the mstore8 opcode can be used to store a single byte at a memory location. Here, we are storing the value `0xff` at the memory location pointed to by `start`, which occupies the 12th byte of the memory slot. 7. With all the values packed into their correct memory locations, we can now call `keccak256` on the memory slot starting at `start`, and pass in the length of the memory slot as the second parameter. This will return a 32 byte hash, which we can truncate to get the final address.

> ℹ️ **Note**
>
> You can check out the complete code for this factory implementation here. Also check out OpenZeppelin's CREATE2 library implementation, which has been used as inspiration for this tutorial. Finally, Forge offers some `CREATE2` address computation helper functions out of the box. Check them out.

Testing our factory

Create a file named `Create2.t.sol` Inside the `test` directory. Initialize a contract named `Create2Test` like this:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.20;

import "forge-std/Test.sol";
import {Counter} from "../src/Counter.sol";
import {Create2} from "../src/Create2.sol";

contract Create2Test is Test {

}
```

Initialize the following state variables and the `setUp()` function:

```
    Create2 internal create2;
    Counter internal counter;

    function setUp() public {
        create2 = new Create2();
        counter = new Counter();
    }
```

Create a new function named `testDeterministicDeploy()` that:

1. Deploys a new instance of the `Create2` contract.
2. Deals out a 100 ETH to the a specific address that we will use to impersonate as the caller for all subsequent calls using the `prank` cheatcode.
3. Sets up the `salt` and `bytecode` params
4. Uses the previously deployed `Create2` contract to deploy the `Counter` contract at a deterministic address.
5. Checks if the contract was deployed at the correct address, by asserting that the computed address is equal to the deployed address.

```
    function testDeterministicDeploy() public {
        vm.deal(address(0x1), 100 ether);

        vm.startPrank(address(0x1));
        bytes32 salt = "12345";
        bytes memory creationCode =
abi.encodePacked(type(Counter).creationCode);

        address computedAddress = create2.computeAddress(salt,
keccak256(creationCode));
        address deployedAddress = create2.deploy(salt , creationCode);
        vm.stopPrank();

        assertEq(computedAddress, deployedAddress);
    }
```

Save all your files, and run the test using `forge test --match-path test/Create2.t.sol -vvvv` . Your test should pass without any errors.

ℹ️ **Note** This guide is intended to help understand `CREATE2` . In most use cases, you won't need to write and use your own deployer, and can use an existing deterministic deployer. In forge scripts, using `new MyContract{salt: salt}()` will use the deterministic deployer at 0x4e59b44847b379578588920ca78fbf26c0b4956c.

Forking Mainnet with Cast and Anvil

Introduction

By combining Anvil and Cast, you can fork and test by interacting with contracts on a real network. The goal of this tutorial is to show you how to transfer Dai tokens from someone who holds Dai to an account created by Anvil.

Set Up

Let's start by forking mainnet.

```
anvil --fork-url https://mainnet.infura.io/v3/$INFURA_KEY
```

You will see 10 accounts are created with their public and private keys. We will work with `0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266` (Let's call this user Alice).

Transferring Dai

Go to Etherscan and search for holders of Dai tokens (here). Let's pick a random account. In this example we will be using `0xfc2eE3bD619B7cfb2dE2C797b96DeeCbD7F68e46`. Let's export our contracts and accounts as environment variables:

```
export ALICE=0xf39fd6e51aad88f6f4ce6ab8827279cfffb92266
export DAI=0x6b175474e89094c44da98b954eedeac495271d0f
export UNLUCKY_USER=0xfc2eE3bD619B7cfb2dE2C797b96DeeCbD7F68e46
```

We can check Alice's balance using `cast call`:

```
$ cast call $DAI \
  "balanceOf(address)(uint256)" \
  $ALICE
0
```

Similarly, we can also check our unlucky user's balance using `cast call`:

```
$ cast call $DAI \
  "balanceOf(address)(uint256)" \
  $UNLUCKY_USER
2184011497352420810932243 8
```

Let's transfer some tokens from the unlucky user to Alice using `cast send`:

```
# This calls Anvil and lets us impersonate our unlucky user
$ cast rpc anvil_impersonateAccount $UNLUCKY_USER
$ cast send $DAI \
--from $UNLUCKY_USER \
  "transfer(address,uint256)(bool)" \
  $ALICE \
  3000000000000000000000000 \
  --unlocked
blockHash
0xbf31c45f6935a0714bb4f709b5e3850ab0cc2f8bffe895fefb653d154e0aa062
blockNumber             15052891
...
```

Let's check that the transfer worked:

```
cast call $DAI \
  "balanceOf(address)(uint256)" \
  $ALICE
3000000000000000000000000

$ cast call $DAI \
  "balanceOf(address)(uint256)" \
  $UNLUCKY_USER
2154011497352420810932243 8
```

Foundry Tutorials (Unofficial) Videos

Unofficial *youtube playlists* of Foundry tutorials from Blockchain educators.

URL	Description	Author
Blockchain Developer, Solidity, Foundry Full Course 2023 ~ Learn Solidity, Blockchain Development, & Smart Contracts Powered By AI - Full Course	Patrick Collins	
Foundry ~ Playlist of beginner level videos on Foundry	Smart Contract Programmer	
	A Complete Introduction to Smart Contract Development With Foundry	Axelar

FAQ

This is a collection of common questions and answers. If you do not find your question listed here, hop in the Telegram support channel and let us help you!

I can't build from source!

Make sure you're on the latest stable Rust toolchain:

```
rustup default stable
rustup update stable
```

`libusb` error when running `forge/cast`

If you are using the binaries as released, you may see the following error on MacOS:

```
dyld: Library not loaded: /usr/local/opt/libusb/lib/libusb-1.0.0.dylib
```

In order to fix this, you must install the `libusb` library:

```
brew install libusb
```

Out of date `GLIBC`

If you run into an error resembling the following after using `foundryup`:

```
forge: /lib/x86_64-linux-gnu/libc.so.6: version 'GLIBC_2.29' not found
(required by forge)
```

There are 2 workarounds:

1. Building from source
2. Using Docker

Help! I can't see my logs!

Forge does not display logs by default. If you want to see logs from Hardhat's `console.log` or from DSTest-style `log_*` events, you need to run `forge test` with verbosity 2 (`-vv`).

If you want to see other events your contracts emit, you need to run with traces enabled. To do that, set the verbosity to 3 (`-vvv`) to see traces for failing tests, or 4 (`-vvvv`) to see traces for all tests.

My tests are failing and I don't know why!

To gain better insight into why your tests are failing, try using traces. To enable traces, you need to increase the verbosity on forge test to at least 3 (`-vvv`) but you can go as high as 5 (`-vvvvv`) for even more traces.

You can learn more about traces in our Understanding Traces chapter.

How do I use `console.log`?

To use Hardhat's `console.log` you must add it to your project by copying the file over from here.

Alternatively, you can use Forge Std which comes bundled with `console.log` . To use `console.log` from Forge Std, you have to import it:

```
import "forge-std/console.sol";
```

How do I run specific tests?

If you want to run only a few tests, you can use `--match-test` to filter test functions, `--match-contract` to filter test contracts, and `--match-path` to filter test files on `forge test` .

How do I use a specific Solidity compiler?

Forge will try to auto-detect what Solidity compiler works for your project.

To use a specific Solidity compiler, you can set `solc` in your config file, or pass `--use solc:<version>` to a Forge command that supports it (e.g. `forge build` or `forge test`). Paths to a solc binary are also accepted. To use a specific local solc binary, you can set `solc = "<path to solc>"` in your config file, or pass `--use "<path to solc>"` . The solc version/path can also be set via the env variable `FOUNDRY_SOLC=<version/path>` , but the cli arg `--use` has priority.

For example, if you have a project that supports all 0.7.x Solidity versions, but you want to compile with solc 0.7.0, you could use `forge build --use solc:0.7.0` .

How do I fork from a live network?

To fork from a live network, pass `--fork-url <URL>` to `forge test`. You can also fork from a specific block using `--fork-block-number <BLOCK>`, which adds determinism to your test, and allows Forge to cache the chain data for that block.

For example, to fork from Ethereum mainnet at block 10,000,000 you could use: `forge test --fork-url $MAINNET_RPC_URL --fork-block-number 10000000`.

How do I add my own assertions?

You can add your own assertions by creating your own base test contract and having that inherit from the test framework of your choice.

For example, if you use DSTest, you could create a base test contract like this:

```
contract TestBase is DSTest {
    function myCustomAssertion(uint a, uint b) {
      if (a != b) {
          emit log_string("a and b did not match");
          fail();
      }
    }
}
```

You would then inherit from `TestBase` in your test contracts.

```
contract MyContractTest is TestBase {
    function testSomething() {
        // ...
    }
}
```

Similarly, if you use Forge Std, you can create a base test contract that inherits from `Test`.

For a good example of a base test contract that has helper methods and custom assertions, see Solmate's `DSTestPlus`.

How do I use Forge offline?

Forge will sometimes check for newer Solidity versions that fit your project. To use Forge offline, use the `--offline` flag.

I'm getting Solc errors

solc-bin doesn't offer static builds for apple silicon. Foundry relies on svm to install native builds for apple silicon.

All solc versions are installed under `~/.svm/`. If you encounter solc related errors, such as `SolcError: ...` please to nuke `~/.svm/` and try again, this will trigger a fresh install and usually resolves the issue.

If you're on apple silicon, please ensure the `z3` theorem prover is installed: `brew install z3`

> **Note**: native apple silicon builds are only available from `0.8.5` upwards. If you need older versions, you must enable apple silicon rosetta to run them.

Forge fails in JavaScript monorepos (`pnpm`)

Managers like `pnpm` use symlinks to manage `node_modules` folders.

A common layout may look like:

```
├── contracts
│   ├── contracts
│   ├── foundry.toml
│   ├── lib
│   ├── node_modules
│   ├── package.json
├── node_modules
│   ├── ...
├── package.json
├── pnpm-lock.yaml
├── pnpm-workspace.yaml
```

Where the Foundry workspace is in `./contracts`, but packages in `./contracts/node_modules` are symlinked to `./node_modules`.

When running `forge build` in `./contracts/node_modules`, this can lead to an error like:

```
error[6275]: ParserError: Source
"node_modules/@openzeppelin/contracts/utils/cryptography/draft-EIP712.sol" not
found: File outside of allowed directories. The following are allowed: "
<repo>/contracts", "<repo>/contracts/contracts", "<repo>/contracts/lib".
 --> node_modules/@openzeppelin/contracts/token/ERC20/extensions/draft-
ERC20Permit.sol:8:1:
  |
8 | import "../../../utils/cryptography/draft-EIP712.sol";
```

This error happens when `solc` was able to resolve symlinked files, but they're outside the Foundry workspace (`./contracts`).

Adding `node_modules` to `allow_paths` in `foundry.toml` grants solc access to that directory, and it will be able to read it:

```
# This translates to `solc --allow-paths ../node_modules`
allow_paths = ["../node_modules"]
```

Note that the path is relative to the Foundry workspace. See also solc allowed-paths

I'm getting `Permission denied (os error 13)`

If you see an error like

```
Failed to create artifact parent folder
"/.../MyProject/out/IsolationModeMagic.sol": Permission denied (os error 13)
```

Then there's likely a folder permission issue. Ensure `user` has write access in the project root's folder.

It has been reported that on linux, canonicalizing paths can result in weird paths (`/_1/...`). This can be resolved by nuking the entire project folder and initializing again.

Connection refused when running `forge build`

If you're unable to access github URLs called by `forge build` , you will see an error like

```
Error:
error sending request for url
(https://raw.githubusercontent.com/roynalnaruto/solc-
builds/ff4ea8a7bbde4488428de69f2c40a7fc56184f5e/macosx/aarch64/list.json):
error trying to connect: tcp connect error: Connection refused (os error 61)
```

Connection failed because access to the URL from your location may be restricted. To solve this, you should set proxy.

You could run `export http_proxy=http://127.0.0.1:7890 https_proxy=http://127.0.0.1:7890` first in the terminal then you will `forge build` successfully.

Contributing

Thanks for your interest in improving Foundry!

There are multiple opportunities to contribute at any level. It doesn't matter if you are just getting started with Rust or are the most weathered expert, we can use your help.

No contribution is too small and all contributions are valued.

This document will help you get started. Do not let the document intimidate you. It should be considered as a guide to help you navigate the process.

The dev Telegram is available for any concerns you may have that are not covered in this guide.

First check the Contribution guidelines

Some articles to walk you through the first steps:

- Make a first contribution
- Add a cheatcode (outdated, see the Foundry dev docs for a shorter but up-to-date tutorial)

References

- CLI Reference
- forge Commands
- cast Commands
- anvil Reference
- chisel Reference
- Config Reference
- Cheatcodes Reference
- Forge Standard Library Reference
- ds-test Reference

CLI Reference

Automatically-generated CLI reference from `--help` output.

- forge
  - forge bind
  - forge build
  - forge cache
    - forge cache clean
    - forge cache ls
  - forge clean
  - forge completions
  - forge config
  - forge coverage
  - forge create
  - forge debug
  - forge doc
  - forge flatten
  - forge fmt
  - forge geiger
  - forge generate
    - forge generate test
  - forge generate-fig-spec
  - forge init
  - forge inspect
  - forge install
  - forge remappings
  - forge remove
  - forge script
  - forge selectors
    - forge selectors collision
    - forge selectors upload
    - forge selectors list
  - forge snapshot
  - forge test
  - forge tree
  - forge update
  - forge verify-check
  - forge verify-contract
- cast

- `cast 4byte`
- `cast 4byte-decode`
- `cast 4byte-event`
- `cast abi-decode`
- `cast abi-encode`
- `cast access-list`
- `cast address-zero`
- `cast admin`
- `cast age`
- `cast balance`
- `cast base-fee`
- `cast bind`
- `cast block`
- `cast block-number`
- `cast call`
  - `cast call --create`
- `cast calldata`
- `cast calldata-decode`
- `cast chain`
- `cast chain-id`
- `cast client`
- `cast code`
- `cast codesize`
- `cast completions`
- `cast compute-address`
- `cast concat-hex`
- `cast create2`
- `cast decode-transaction`
- `cast disassemble`
- `cast estimate`
  - `cast estimate --create`
- `cast etherscan-source`
- `cast find-block`
- `cast format-bytes32-string`
- `cast from-bin`
- `cast from-fixed-point`
- `cast from-rlp`
- `cast from-utf8`
- `cast from-wei`
- `cast gas-price`
- `cast generate-fig-spec`
- `cast hash-zero`
- `cast implementation`

- `cast index`
- `cast interface`
- `cast keccak`
- `cast logs`
- `cast lookup-address`
- `cast max-int`
- `cast max-uint`
- `cast min-int`
- `cast mktx`
  - `cast mktx --create`
- `cast namehash`
- `cast nonce`
- `cast parse-bytes32-address`
- `cast parse-bytes32-string`
- `cast pretty-calldata`
- `cast proof`
- `cast publish`
- `cast receipt`
- `cast resolve-name`
- `cast rpc`
- `cast run`
- `cast selectors`
- `cast send`
  - `cast send --create`
- `cast shl`
- `cast shr`
- `cast sig`
- `cast sig-event`
- `cast storage`
- `cast to-ascii`
- `cast to-base`
- `cast to-bytes32`
- `cast to-check-sum-address`
- `cast to-dec`
- `cast to-fixed-point`
- `cast to-hex`
- `cast to-hexdata`
- `cast to-int256`
- `cast to-rlp`
- `cast to-uint256`
- `cast to-unit`
- `cast to-wei`
- `cast tx`

    - `cast upload-signature`
    - `cast wallet`
        - `cast wallet new`
        - `cast wallet new-mnemonic`
        - `cast wallet vanity`
        - `cast wallet address`
        - `cast wallet sign`
        - `cast wallet verify`
        - `cast wallet import`
        - `cast wallet list`
        - `cast wallet derive-private-key`

- `anvil`
    - `anvil completions`
    - `anvil generate-fig-spec`
- `chisel`
    - `chisel list`
    - `chisel load`
    - `chisel view`
    - `chisel clear-cache`

forge

Build, test, fuzz, debug and deploy Solidity contracts

```
$ forge --help
Usage: forge <COMMAND>

Commands:
  bind                Generate Rust bindings for smart contracts
  build               Build the project's smart contracts [aliases: b, compile]
  cache               Manage the Foundry cache
  clean               Remove the build artifacts and cache directories [aliases:
cl]
  completions         Generate shell completions script [aliases: com]
  config              Display the current config [aliases: co]
  coverage            Generate coverage reports
  create              Deploy a smart contract [aliases: c]
  debug               Debugs a single smart contract as a script [aliases: d]
  doc                 Generate documentation for the project
  flatten             Flatten a source file and all of its imports into one file
[aliases: f]
  fmt                 Format Solidity source files
  geiger              Detects usage of unsafe cheat codes in a project and its
dependencies
  generate            Generate scaffold files
  generate-fig-spec   Generate Fig autocompletion spec [aliases: fig]
  help                Print this message or the help of the given subcommand(s)
  init                Create a new Forge project
  inspect             Get specialized information about a smart contract
[aliases: in]
  install             Install one or multiple dependencies [aliases: i]
  remappings          Get the automatically inferred remappings for the project
[aliases: re]
  remove              Remove one or multiple dependencies [aliases: rm]
  script              Run a smart contract as a script, building transactions
that can be sent
                          onchain
  selectors           Function selector utilities [aliases: se]
  snapshot            Create a snapshot of each test's gas usage [aliases: s]
  test                Run the project's tests [aliases: t]
  tree                Display a tree visualization of the project's dependency
graph [aliases:
                          tr]
  update              Update one or multiple dependencies [aliases: u]
  verify-check        Check verification status on Etherscan [aliases: vc]
  verify-contract     Verify smart contracts on Etherscan [aliases: v]

Options:
  -h, --help     Print help
  -V, --version  Print version

Find more information in the book:
http://book.getfoundry.sh/reference/forge/forge.html
```

forge bind

Generate Rust bindings for smart contracts

```
$ forge bind --help
Usage: forge bind [OPTIONS]

Options:
  -b, --bindings-path <PATH>
          Path to where the contract artifacts are stored

      --select <SELECT>
          Create bindings only for contracts whose names match the specified
filter(s)

      --skip <SKIP>
          Create bindings only for contracts whose names do not match the
specified filter(s)

      --select-all
          Explicitly generate bindings for all contracts

          By default all contracts ending with `Test` or `Script` are excluded.

      --crate-name <NAME>
          The name of the Rust crate to generate.

          This should be a valid crates.io crate name, however, this is not
currently validated by
          this command.

          [default: foundry-contracts]

      --crate-version <VERSION>
          The version of the Rust crate to generate.

          This should be a standard semver version string, however, this is not
currently validated
          by this command.

          [default: 0.1.0]

      --module
          Generate the bindings as a module instead of a crate

      --overwrite
          Overwrite existing generated bindings.

          By default, the command will check that the bindings are correct, and
then exit. If
          --overwrite is passed, it will instead delete and overwrite the
bindings.

      --single-file
          Generate bindings as a single file

      --skip-cargo-toml
          Skip Cargo.toml consistency checks

      --skip-build
          Skips running forge build before generating binding
```

```
      --skip-extra-derives
          Don't add any additional derives to generated bindings

  -h, --help
          Print help (see a summary with '-h')

Cache options:
      --force
          Clear the cache and artifacts folder and recompile

Build options:
      --no-cache
          Disable the cache

Linker options:
      --libraries <LIBRARIES>
          Set pre-linked libraries

          [env: DAPP_LIBRARIES=]

Compiler options:
      --ignored-error-codes <ERROR_CODES>
          Ignore solc warnings by error code

      --deny-warnings
          Warnings will trigger a compiler error

      --no-auto-detect
          Do not auto-detect the `solc` version

      --use <SOLC_VERSION>
          Specify the solc version, or a path to a local solc, to build with.

          Valid values are in the format `x.y.z`, `solc:x.y.z` or
`path/to/solc`.

      --offline
          Do not access the network.

          Missing solc versions will not be installed.

      --via-ir
          Use the Yul intermediate representation compilation pipeline

      --silent
          Don't print anything on startup

      --ast
          Includes the AST as JSON in the compiler output

      --evm-version <VERSION>
          The target EVM version

      --optimize
          Activate the Solidity optimizer

      --optimizer-runs <RUNS>
```

```
            The number of optimizer runs

        --extra-output <SELECTOR>...
            Extra output to include in the contract's artifact.

            Example keys: evm.assembly, ewasm, ir, irOptimized, metadata

            For a full description, see
            https://docs.soliditylang.org/en/v0.8.13/using-the-
compiler.html#input-description

        --extra-output-files <SELECTOR>...
            Extra output to write to separate files.

            Valid values: metadata, ir, irOptimized, ewasm, evm.assembly

Project options:
  -o, --out <PATH>
            The path to the contract artifacts folder

        --revert-strings <REVERT>
            Revert string configuration.

            Possible values are "default", "strip" (remove), "debug" (Solidity-
generated revert
            strings) and "verboseDebug"

        --build-info
            Generate build info files

        --build-info-path <PATH>
            Output path to directory that build info files will be written to

        --root <PATH>
            The project's root path.

            By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
            The contracts source directory

  -R, --remappings <REMAPPINGS>
            The project's remappings

        --remappings-env <ENV>
            The project's remappings from the environment

        --cache-path <PATH>
            The path to the compiler cache

        --lib-paths <PATH>
            The path to the library folder

        --hardhat
            Use the Hardhat-style project layout.

            This is the same as using: `--contracts contracts --lib-paths
```

```
node_modules`.

          [aliases: hh]

      --config-path <FILE>
          Path to the config file
```

forge build

Build the project's smart contracts

```
$ forge build --help
Usage: forge build [OPTIONS]

Options:
  -h, --help
          Print help (see a summary with '-h')

Build options:
      --names
          Print compiled contract names

      --sizes
          Print compiled contract sizes

      --skip <SKIP>...
          Skip building files whose names contain the given filter.

          `test` and `script` are aliases for `.t.sol` and `.s.sol`.

      --no-cache
          Disable the cache

Cache options:
      --force
          Clear the cache and artifacts folder and recompile

Linker options:
      --libraries <LIBRARIES>
          Set pre-linked libraries

          [env: DAPP_LIBRARIES=]

Compiler options:
      --ignored-error-codes <ERROR_CODES>
          Ignore solc warnings by error code

      --deny-warnings
          Warnings will trigger a compiler error

      --no-auto-detect
          Do not auto-detect the `solc` version

      --use <SOLC_VERSION>
          Specify the solc version, or a path to a local solc, to build with.

          Valid values are in the format `x.y.z`, `solc:x.y.z` or
`path/to/solc`.

      --offline
          Do not access the network.

          Missing solc versions will not be installed.

      --via-ir
          Use the Yul intermediate representation compilation pipeline

      --silent
```

```
            Don't print anything on startup

        --ast
            Includes the AST as JSON in the compiler output

        --evm-version <VERSION>
            The target EVM version

        --optimize
            Activate the Solidity optimizer

        --optimizer-runs <RUNS>
            The number of optimizer runs

        --extra-output <SELECTOR>...
            Extra output to include in the contract's artifact.

            Example keys: evm.assembly, ewasm, ir, irOptimized, metadata

            For a full description, see
            https://docs.soliditylang.org/en/v0.8.13/using-the-
compiler.html#input-description

        --extra-output-files <SELECTOR>...
            Extra output to write to separate files.

            Valid values: metadata, ir, irOptimized, ewasm, evm.assembly

Project options:
  -o, --out <PATH>
            The path to the contract artifacts folder

        --revert-strings <REVERT>
            Revert string configuration.

            Possible values are "default", "strip" (remove), "debug" (Solidity-
generated revert
            strings) and "verboseDebug"

        --build-info
            Generate build info files

        --build-info-path <PATH>
            Output path to directory that build info files will be written to

        --root <PATH>
            The project's root path.

            By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
            The contracts source directory

  -R, --remappings <REMAPPINGS>
            The project's remappings

        --remappings-env <ENV>
```

```
          The project's remappings from the environment

      --cache-path <PATH>
          The path to the compiler cache

      --lib-paths <PATH>
          The path to the library folder

      --hardhat
          Use the Hardhat-style project layout.

          This is the same as using: `--contracts contracts --lib-paths
node_modules`.

          [aliases: hh]

      --config-path <FILE>
          Path to the config file

Watch options:
  -w, --watch [<PATH>...]
          Watch the given files or directories for changes.

          If no paths are provided, the source and test directories of the
project are watched.

      --no-restart
          Do not restart the command while it's still running

      --run-all
          Explicitly re-run all tests when a change is made.

          By default, only the tests of the last modified test file are
executed.

      --watch-delay <DELAY>
          File update debounce delay.

          During the delay, incoming change events are accumulated and only
once the delay has
          passed, is an action taken. Note that this does not mean a command
will be started: if
          --no-restart is given and a command is already running, the outcome
of the action will be
          to do nothing.

          Defaults to 50ms. Parses as decimal seconds by default, but using an
integer with the `ms`
          suffix may be more convenient.

          When using --poll mode, you'll want a larger duration, or risk
overloading disk I/O.

      --format-json
          Output the compilation errors in the json format. This is useful when
you want to use the
          output in other tools
```

forge cache

Manage the Foundry cache

```
$ forge cache --help
Usage: forge cache <COMMAND>

Commands:
  clean  Cleans cached data from the global foundry directory
  ls     Shows cached data from the global foundry directory
  help   Print this message or the help of the given subcommand(s)

Options:
  -h, --help  Print help
```

forge cache clean

Cleans cached data from the global foundry directory

```
$ forge cache clean --help
Usage: forge cache clean [OPTIONS] [CHAINS]...

Arguments:
  [CHAINS]...
          The chains to clean the cache for.

          Can also be "all" to clean all chains.

          [env: CHAIN=]
          [default: all]

Options:
  -b, --blocks <BLOCKS>...
          The blocks to clean the cache for

      --etherscan
          Whether to clean the Etherscan cache

  -h, --help
          Print help (see a summary with '-h')
```

forge cache ls

Shows cached data from the global foundry directory

```
$ forge cache ls --help
Usage: forge cache ls [CHAINS]...

Arguments:
  [CHAINS]...
          The chains to list the cache for.

          Can also be "all" to list all chains.

          [env: CHAIN=]
          [default: all]

Options:
  -h, --help
          Print help (see a summary with '-h')
```

forge clean

Remove the build artifacts and cache directories

```
$ forge clean --help
Usage: forge clean [OPTIONS]

Options:
      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

  -h, --help
          Print help (see a summary with '-h')
```

forge completions

Generate shell completions script

```
$ forge completions --help
Usage: forge completions <SHELL>

Arguments:
  <SHELL>  [possible values: bash, elvish, fish, powershell, zsh]

Options:
  -h, --help  Print help
```

forge config

Display the current config

```
$ forge config --help
Usage: forge config [OPTIONS]

Options:
      --basic
          Print only a basic set of the currently set config values

      --json
          Print currently set config values as JSON

      --fix
          Attempt to fix any configuration warnings

  -h, --help
          Print help (see a summary with '-h')

Build options:
      --names
          Print compiled contract names

      --sizes
          Print compiled contract sizes

      --skip <SKIP>...
          Skip building files whose names contain the given filter.

          `test` and `script` are aliases for `.t.sol` and `.s.sol`.

      --no-cache
          Disable the cache

Cache options:
      --force
          Clear the cache and artifacts folder and recompile

Linker options:
      --libraries <LIBRARIES>
          Set pre-linked libraries

          [env: DAPP_LIBRARIES=]

Compiler options:
      --ignored-error-codes <ERROR_CODES>
          Ignore solc warnings by error code

      --deny-warnings
          Warnings will trigger a compiler error

      --no-auto-detect
          Do not auto-detect the `solc` version

      --use <SOLC_VERSION>
          Specify the solc version, or a path to a local solc, to build with.

          Valid values are in the format `x.y.z`, `solc:x.y.z` or
`path/to/solc`.
```

```
      --offline
          Do not access the network.

          Missing solc versions will not be installed.

      --via-ir
          Use the Yul intermediate representation compilation pipeline

      --silent
          Don't print anything on startup

      --ast
          Includes the AST as JSON in the compiler output

      --evm-version <VERSION>
          The target EVM version

      --optimize
          Activate the Solidity optimizer

      --optimizer-runs <RUNS>
          The number of optimizer runs

      --extra-output <SELECTOR>...
          Extra output to include in the contract's artifact.

          Example keys: evm.assembly, ewasm, ir, irOptimized, metadata

          For a full description, see
          https://docs.soliditylang.org/en/v0.8.13/using-the-
compiler.html#input-description

      --extra-output-files <SELECTOR>...
          Extra output to write to separate files.

          Valid values: metadata, ir, irOptimized, ewasm, evm.assembly

Project options:
  -o, --out <PATH>
          The path to the contract artifacts folder

      --revert-strings <REVERT>
          Revert string configuration.

          Possible values are "default", "strip" (remove), "debug" (Solidity-
generated revert
          strings) and "verboseDebug"

      --build-info
          Generate build info files

      --build-info-path <PATH>
          Output path to directory that build info files will be written to

      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
```

```
working directory.

  -C, --contracts <PATH>
          The contracts source directory

  -R, --remappings <REMAPPINGS>
          The project's remappings

      --remappings-env <ENV>
          The project's remappings from the environment

      --cache-path <PATH>
          The path to the compiler cache

      --lib-paths <PATH>
          The path to the library folder

      --hardhat
          Use the Hardhat-style project layout.

          This is the same as using: `--contracts contracts --lib-paths
node_modules`.

          [aliases: hh]

      --config-path <FILE>
          Path to the config file

Watch options:
  -w, --watch [<PATH>...]
          Watch the given files or directories for changes.

          If no paths are provided, the source and test directories of the
project are watched.

      --no-restart
          Do not restart the command while it's still running

      --run-all
          Explicitly re-run all tests when a change is made.

          By default, only the tests of the last modified test file are
executed.

      --watch-delay <DELAY>
          File update debounce delay.

          During the delay, incoming change events are accumulated and only
once the delay has
          passed, is an action taken. Note that this does not mean a command
will be started: if
          --no-restart is given and a command is already running, the outcome
of the action will be
          to do nothing.

          Defaults to 50ms. Parses as decimal seconds by default, but using an
integer with the `ms`
          suffix may be more convenient.
```

```
          When using --poll mode, you'll want a larger duration, or risk
overloading disk I/O.

      --format-json
          Output the compilation errors in the json format. This is useful when
you want to use the
          output in other tools

EVM options:
  -f, --fork-url <URL>
          Fetch state over a remote endpoint instead of starting from an empty
state.

          If you want to fetch state from a specific block number, see --fork-
block-number.

          [aliases: rpc-url]

      --fork-block-number <BLOCK>
          Fetch state from a specific block number over a remote endpoint.

          See --fork-url.

      --fork-retries <RETRIES>
          Number of retries.

          See --fork-url.

      --fork-retry-backoff <BACKOFF>
          Initial retry backoff on encountering errors.

          See --fork-url.

      --no-storage-caching
          Explicitly disables the use of RPC caching.

          All storage slots are read entirely from the endpoint.

          This flag overrides the project's configuration file.

          See --fork-url.

      --initial-balance <BALANCE>
          The initial balance of deployed test contracts

      --sender <ADDRESS>
          The address which will be executing tests

      --ffi
          Enable the FFI cheatcode

      --always-use-create-2-factory
          Use the create 2 factory in all cases including tests and non-
broadcasting scripts

  -v, --verbosity...
          Verbosity of the EVM.
```

```
          Pass multiple times to increase the verbosity (e.g. -v, -vv, -vvv).

          Verbosity levels:
          - 2: Print logs for all tests
          - 3: Print execution traces for failing tests
          - 4: Print execution traces for all tests, and setup traces for
failing tests
          - 5: Print execution and setup traces for all tests

Fork config:
      --compute-units-per-second <CUPS>
          Sets the number of assumed available compute units per second for
this provider

          default value: 330

          See also --fork-url and
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

      --no-rpc-rate-limit
          Disables rate limiting for this node's provider.

          See also --fork-url and
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

          [aliases: no-rate-limit]

Executor environment config:
      --gas-limit <GAS_LIMIT>
          The block gas limit

      --code-size-limit <CODE_SIZE>
          EIP-170: Contract code size limit in bytes. Useful to increase this
because of tests. By
          default, it is 0x6000 (~25kb)

      --chain <CHAIN>
          The chain name or EIP-155 chain ID

          [aliases: chain-id]

      --gas-price <GAS_PRICE>
          The gas price

      --block-base-fee-per-gas <FEE>
          The base fee in a block

          [aliases: base-fee]

      --tx-origin <ADDRESS>
          The transaction origin

      --block-coinbase <ADDRESS>
          The coinbase of the block
```

```
      --block-timestamp <TIMESTAMP>
          The timestamp of the block

      --block-number <BLOCK>
          The block number

      --block-difficulty <DIFFICULTY>
          The block difficulty

      --block-prevrandao <PREVRANDAO>
          The block prevrandao value. NOTE: Before merge this field was
mix_hash

      --block-gas-limit <GAS_LIMIT>
          The block gas limit

      --memory-limit <MEMORY_LIMIT>
          The memory limit per EVM execution in bytes. If this limit is
exceeded, a `MemoryLimitOOG`
          result is thrown.

          The default is 128MiB.

      --disable-block-gas-limit
          Whether to disable the block gas limit checks

          [aliases: no-gas-limit]

      --isolate
          Whether to enable isolation of calls. In isolation mode all top-level
calls are executed
          as a separate transaction in a separate EVM context, enabling more
precise gas accounting
          and transaction state changes
```

forge coverage

Generate coverage reports

```
$ forge coverage --help
Usage: forge coverage [OPTIONS]

Options:
      --report <REPORT>
          The report type to use for coverage.

          This flag can be used multiple times.

          [default: summary]
          [possible values: summary, lcov, debug, bytecode]

      --ir-minimum
          Enable viaIR with minimum optimization

          This can fix most of the "stack too deep" errors while resulting a
relatively accurate
          source map.

  -r, --report-file <PATH>
          The path to output the report.

          If not specified, the report will be stored in the root of the
project.

  -h, --help
          Print help (see a summary with '-h')

Test filtering:
      --match-test <REGEX>
          Only run test functions matching the specified regex pattern

          [aliases: mt]

      --no-match-test <REGEX>
          Only run test functions that do not match the specified regex pattern

          [aliases: nmt]

      --match-contract <REGEX>
          Only run tests in contracts matching the specified regex pattern

          [aliases: mc]

      --no-match-contract <REGEX>
          Only run tests in contracts that do not match the specified regex
pattern

          [aliases: nmc]

      --match-path <GLOB>
          Only run tests in source files matching the specified glob pattern

          [aliases: mp]

      --no-match-path <GLOB>
          Only run tests in source files that do not match the specified glob
```

```
pattern

          [aliases: nmp]

EVM options:
  -f, --fork-url <URL>
          Fetch state over a remote endpoint instead of starting from an empty
state.

          If you want to fetch state from a specific block number, see --fork-
block-number.

          [aliases: rpc-url]

      --fork-block-number <BLOCK>
          Fetch state from a specific block number over a remote endpoint.

          See --fork-url.

      --fork-retries <RETRIES>
          Number of retries.

          See --fork-url.

      --fork-retry-backoff <BACKOFF>
          Initial retry backoff on encountering errors.

          See --fork-url.

      --no-storage-caching
          Explicitly disables the use of RPC caching.

          All storage slots are read entirely from the endpoint.

          This flag overrides the project's configuration file.

          See --fork-url.

      --initial-balance <BALANCE>
          The initial balance of deployed test contracts

      --sender <ADDRESS>
          The address which will be executing tests

      --ffi
          Enable the FFI cheatcode

      --always-use-create-2-factory
          Use the create 2 factory in all cases including tests and non-
broadcasting scripts

  -v, --verbosity...
          Verbosity of the EVM.

          Pass multiple times to increase the verbosity (e.g. -v, -vv, -vvv).

          Verbosity levels:
          - 2: Print logs for all tests
```

```
          - 3: Print execution traces for failing tests
          - 4: Print execution traces for all tests, and setup traces for
failing tests
          - 5: Print execution and setup traces for all tests

Fork config:
      --compute-units-per-second <CUPS>
          Sets the number of assumed available compute units per second for
this provider

          default value: 330

          See also --fork-url and
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

      --no-rpc-rate-limit
          Disables rate limiting for this node's provider.

          See also --fork-url and
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

          [aliases: no-rate-limit]

Executor environment config:
      --gas-limit <GAS_LIMIT>
          The block gas limit

      --code-size-limit <CODE_SIZE>
          EIP-170: Contract code size limit in bytes. Useful to increase this
because of tests. By
          default, it is 0x6000 (~25kb)

      --chain <CHAIN>
          The chain name or EIP-155 chain ID

          [aliases: chain-id]

      --gas-price <GAS_PRICE>
          The gas price

      --block-base-fee-per-gas <FEE>
          The base fee in a block

          [aliases: base-fee]

      --tx-origin <ADDRESS>
          The transaction origin

      --block-coinbase <ADDRESS>
          The coinbase of the block

      --block-timestamp <TIMESTAMP>
          The timestamp of the block

      --block-number <BLOCK>
          The block number
```

```
      --block-difficulty <DIFFICULTY>
          The block difficulty

      --block-prevrandao <PREVRANDAO>
          The block prevrandao value. NOTE: Before merge this field was
mix_hash

      --block-gas-limit <GAS_LIMIT>
          The block gas limit

      --memory-limit <MEMORY_LIMIT>
          The memory limit per EVM execution in bytes. If this limit is
exceeded, a `MemoryLimitOOG`
          result is thrown.

          The default is 128MiB.

      --disable-block-gas-limit
          Whether to disable the block gas limit checks

          [aliases: no-gas-limit]

      --isolate
          Whether to enable isolation of calls. In isolation mode all top-level
calls are executed
          as a separate transaction in a separate EVM context, enabling more
precise gas accounting
          and transaction state changes

Cache options:
      --force
          Clear the cache and artifacts folder and recompile

Build options:
      --no-cache
          Disable the cache

Linker options:
      --libraries <LIBRARIES>
          Set pre-linked libraries

          [env: DAPP_LIBRARIES=]

Compiler options:
      --ignored-error-codes <ERROR_CODES>
          Ignore solc warnings by error code

      --deny-warnings
          Warnings will trigger a compiler error

      --no-auto-detect
          Do not auto-detect the `solc` version

      --use <SOLC_VERSION>
          Specify the solc version, or a path to a local solc, to build with.

          Valid values are in the format `x.y.z`, `solc:x.y.z` or
```

```
`path/to/solc`.

      --offline
          Do not access the network.

          Missing solc versions will not be installed.

      --via-ir
          Use the Yul intermediate representation compilation pipeline

      --silent
          Don't print anything on startup

      --ast
          Includes the AST as JSON in the compiler output

      --evm-version <VERSION>
          The target EVM version

      --optimize
          Activate the Solidity optimizer

      --optimizer-runs <RUNS>
          The number of optimizer runs

      --extra-output <SELECTOR>...
          Extra output to include in the contract's artifact.

          Example keys: evm.assembly, ewasm, ir, irOptimized, metadata

          For a full description, see
          https://docs.soliditylang.org/en/v0.8.13/using-the-
compiler.html#input-description

      --extra-output-files <SELECTOR>...
          Extra output to write to separate files.

          Valid values: metadata, ir, irOptimized, ewasm, evm.assembly

Project options:
  -o, --out <PATH>
          The path to the contract artifacts folder

      --revert-strings <REVERT>
          Revert string configuration.

          Possible values are "default", "strip" (remove), "debug" (Solidity-
generated revert
          strings) and "verboseDebug"

      --build-info
          Generate build info files

      --build-info-path <PATH>
          Output path to directory that build info files will be written to

      --root <PATH>
          The project's root path.
```

```
          By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
          The contracts source directory

  -R, --remappings <REMAPPINGS>
          The project's remappings

      --remappings-env <ENV>
          The project's remappings from the environment

      --cache-path <PATH>
          The path to the compiler cache

      --lib-paths <PATH>
          The path to the library folder

      --hardhat
          Use the Hardhat-style project layout.

          This is the same as using: `--contracts contracts --lib-paths
node_modules`.

          [aliases: hh]

      --config-path <FILE>
          Path to the config file
```

forge create

Deploy a smart contract

```
$ forge create --help
Usage: forge create [OPTIONS] <CONTRACT>

Arguments:
  <CONTRACT>
          The contract identifier in the form `<path>:<contractname>`

Options:
      --constructor-args <ARGS>...
          The constructor arguments

      --constructor-args-path <PATH>
          The path to a file containing the constructor arguments

      --verify
          Verify contract after creation

      --unlocked
          Send via `eth_sendTransaction` using the `--from` argument or
`$ETH_FROM` as sender

      --show-standard-json-input
          Prints the standard json compiler input if `--verify` is provided.

          The standard json compiler input can be used to manually submit
contract verification in
          the browser.

  -h, --help
          Print help (see a summary with '-h')

Display options:
      --json
          Print the deployment information as JSON

Cache options:
      --force
          Clear the cache and artifacts folder and recompile

Build options:
      --no-cache
          Disable the cache

Linker options:
      --libraries <LIBRARIES>
          Set pre-linked libraries

          [env: DAPP_LIBRARIES=]

Compiler options:
      --ignored-error-codes <ERROR_CODES>
          Ignore solc warnings by error code

      --deny-warnings
          Warnings will trigger a compiler error

      --no-auto-detect
```

```
            Do not auto-detect the `solc` version

      --use <SOLC_VERSION>
          Specify the solc version, or a path to a local solc, to build with.

          Valid values are in the format `x.y.z`, `solc:x.y.z` or
`path/to/solc`.

      --offline
          Do not access the network.

          Missing solc versions will not be installed.

      --via-ir
          Use the Yul intermediate representation compilation pipeline

      --silent
          Don't print anything on startup

      --ast
          Includes the AST as JSON in the compiler output

      --evm-version <VERSION>
          The target EVM version

      --optimize
          Activate the Solidity optimizer

      --optimizer-runs <RUNS>
          The number of optimizer runs

      --extra-output <SELECTOR>...
          Extra output to include in the contract's artifact.

          Example keys: evm.assembly, ewasm, ir, irOptimized, metadata

          For a full description, see
          https://docs.soliditylang.org/en/v0.8.13/using-the-
compiler.html#input-description

      --extra-output-files <SELECTOR>...
          Extra output to write to separate files.

          Valid values: metadata, ir, irOptimized, ewasm, evm.assembly

Project options:
  -o, --out <PATH>
          The path to the contract artifacts folder

      --revert-strings <REVERT>
          Revert string configuration.

          Possible values are "default", "strip" (remove), "debug" (Solidity-
generated revert
          strings) and "verboseDebug"

      --build-info
          Generate build info files
```

```
      --build-info-path <PATH>
          Output path to directory that build info files will be written to

      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
          The contracts source directory

  -R, --remappings <REMAPPINGS>
          The project's remappings

      --remappings-env <ENV>
          The project's remappings from the environment

      --cache-path <PATH>
          The path to the compiler cache

      --lib-paths <PATH>
          The path to the library folder

      --hardhat
          Use the Hardhat-style project layout.

          This is the same as using: `--contracts contracts --lib-paths
node_modules`.

          [aliases: hh]

      --config-path <FILE>
          Path to the config file

Transaction options:
      --gas-limit <GAS_LIMIT>
          Gas limit for the transaction

          [env: ETH_GAS_LIMIT=]

      --gas-price <PRICE>
          Gas price for legacy transactions, or max fee per gas for EIP1559
transactions

          [env: ETH_GAS_PRICE=]

      --priority-gas-price <PRICE>
          Max priority fee per gas for EIP1559 transactions

          [env: ETH_PRIORITY_GAS_PRICE=]

      --value <VALUE>
          Ether to send in the transaction, either specified in wei, or as a
string with a unit
          type.
```

```
            Examples: 1ether, 10gwei, 0.01ether

        --nonce <NONCE>
            Nonce for the transaction

        --legacy
            Send a legacy transaction instead of an EIP1559 transaction.

            This is automatically enabled for common networks without EIP1559.

    Ethereum options:
      -r, --rpc-url <URL>
            The RPC endpoint

            [env: ETH_RPC_URL=]

        --flashbots
            Use the Flashbots RPC URL with fast mode
    (https://rpc.flashbots.net/fast). This shares the
            transaction privately with all registered builders.
            https://docs.flashbots.net/flashbots-protect/quick-start#faster-
    transactions

        --jwt-secret <JWT_SECRET>
            JWT Secret for the RPC endpoint.

            The JWT secret will be used to create a JWT for a RPC. For example,
    the following can be
            used to simulate a CL `engine_forkchoiceUpdated` call:

            cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

    '["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
            "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

    "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

            [env: ETH_RPC_JWT_SECRET=]

      -e, --etherscan-api-key <KEY>
            The Etherscan (or equivalent) API key

            [env: ETHERSCAN_API_KEY=]

      -c, --chain <CHAIN>
            The chain name or EIP-155 chain ID

            [env: CHAIN=]

    Wallet options - raw:
      -f, --from <ADDRESS>
            The sender account

            [env: ETH_FROM=]

      -i, --interactive
            Open an interactive prompt to enter your private key
```

```
        --private-key <RAW_PRIVATE_KEY>
            Use the provided private key

        --mnemonic <MNEMONIC>
            Use the mnemonic phrase of mnemonic file at the specified path

        --mnemonic-passphrase <PASSPHRASE>
            Use a BIP39 passphrase for the mnemonic

        --mnemonic-derivation-path <PATH>
            The wallet derivation path.

            Works with both --mnemonic-path and hardware wallets.

        --mnemonic-index <INDEX>
            Use the private key from the given mnemonic index.

            Used with --mnemonic-path.

            [default: 0]

        --retries <RETRIES>
            Number of attempts for retrying verification

            [default: 5]

        --delay <DELAY>
            Optional delay to apply inbetween verification attempts, in seconds

            [default: 5]

    Wallet options - keystore:
        --keystore <PATH>
            Use the keystore in the given folder or file

            [env: ETH_KEYSTORE=]

        --account <ACCOUNT_NAME>
            Use a keystore from the default keystores folder
    (~/.foundry/keystores) by its filename

            [env: ETH_KEYSTORE_ACCOUNT=]

        --password <PASSWORD>
            The keystore password.

            Used with --keystore.

        --password-file <PASSWORD_FILE>
            The keystore password file path.

            Used with --keystore.

            [env: ETH_PASSWORD=]

    Wallet options - hardware wallet:
      -l, --ledger
            Use a Ledger hardware wallet
```

```
  -t, --trezor
          Use a Trezor hardware wallet

Wallet options - AWS KMS:
      --aws
          Use AWS Key Management Service

Verifier options:
      --verifier <VERIFIER>
          The contract verification provider to use

          [default: etherscan]
          [possible values: etherscan, sourcify, blockscout]

      --verifier-url <VERIFIER_URL>
          The verifier URL, if using a custom provider

          [env: VERIFIER_URL=]
```

forge debug

Debugs a single smart contract as a script

```
$ forge debug --help
Usage: forge debug [OPTIONS] <PATH> [ARGS]...

Arguments:
  <PATH>
          The contract you want to run. Either the file path or contract name.

          If multiple contracts exist in the same file you must specify the
target contract with
          --target-contract.

  [ARGS]...
          Arguments to pass to the script function

Options:
      --target-contract <CONTRACT_NAME>
          The name of the contract you want to run

          [aliases: tc]

  -s, --sig <SIGNATURE>
          The signature of the function you want to call in the contract, or
raw calldata

          [default: run()]

      --debug
          Open the script in the debugger

  -h, --help
          Print help (see a summary with '-h')

Cache options:
      --force
          Clear the cache and artifacts folder and recompile

Build options:
      --no-cache
          Disable the cache

Linker options:
      --libraries <LIBRARIES>
          Set pre-linked libraries

          [env: DAPP_LIBRARIES=]

Compiler options:
      --ignored-error-codes <ERROR_CODES>
          Ignore solc warnings by error code

      --deny-warnings
          Warnings will trigger a compiler error

      --no-auto-detect
          Do not auto-detect the `solc` version

      --use <SOLC_VERSION>
```

```
            Specify the solc version, or a path to a local solc, to build with.

            Valid values are in the format `x.y.z`, `solc:x.y.z` or
`path/to/solc`.

      --offline
            Do not access the network.

            Missing solc versions will not be installed.

      --via-ir
            Use the Yul intermediate representation compilation pipeline

      --silent
            Don't print anything on startup

      --ast
            Includes the AST as JSON in the compiler output

      --evm-version <VERSION>
            The target EVM version

      --optimize
            Activate the Solidity optimizer

      --optimizer-runs <RUNS>
            The number of optimizer runs

      --extra-output <SELECTOR>...
            Extra output to include in the contract's artifact.

            Example keys: evm.assembly, ewasm, ir, irOptimized, metadata

            For a full description, see
            https://docs.soliditylang.org/en/v0.8.13/using-the-
compiler.html#input-description

      --extra-output-files <SELECTOR>...
            Extra output to write to separate files.

            Valid values: metadata, ir, irOptimized, ewasm, evm.assembly

Project options:
  -o, --out <PATH>
            The path to the contract artifacts folder

      --revert-strings <REVERT>
            Revert string configuration.

            Possible values are "default", "strip" (remove), "debug" (Solidity-
generated revert
            strings) and "verboseDebug"

      --build-info
            Generate build info files

      --build-info-path <PATH>
            Output path to directory that build info files will be written to
```

```
      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
          The contracts source directory

  -R, --remappings <REMAPPINGS>
          The project's remappings

      --remappings-env <ENV>
          The project's remappings from the environment

      --cache-path <PATH>
          The path to the compiler cache

      --lib-paths <PATH>
          The path to the library folder

      --hardhat
          Use the Hardhat-style project layout.

          This is the same as using: `--contracts contracts --lib-paths
node_modules`.

          [aliases: hh]

      --config-path <FILE>
          Path to the config file

EVM options:
  -f, --fork-url <URL>
          Fetch state over a remote endpoint instead of starting from an empty
state.

          If you want to fetch state from a specific block number, see --fork-
block-number.

          [aliases: rpc-url]

      --fork-block-number <BLOCK>
          Fetch state from a specific block number over a remote endpoint.

          See --fork-url.

      --fork-retries <RETRIES>
          Number of retries.

          See --fork-url.

      --fork-retry-backoff <BACKOFF>
          Initial retry backoff on encountering errors.

          See --fork-url.
```

```
      --no-storage-caching
          Explicitly disables the use of RPC caching.

          All storage slots are read entirely from the endpoint.

          This flag overrides the project's configuration file.

          See --fork-url.

      --initial-balance <BALANCE>
          The initial balance of deployed test contracts

      --sender <ADDRESS>
          The address which will be executing tests

      --ffi
          Enable the FFI cheatcode

      --always-use-create-2-factory
          Use the create 2 factory in all cases including tests and non-
broadcasting scripts

  -v, --verbosity...
          Verbosity of the EVM.

          Pass multiple times to increase the verbosity (e.g. -v, -vv, -vvv).

          Verbosity levels:
          - 2: Print logs for all tests
          - 3: Print execution traces for failing tests
          - 4: Print execution traces for all tests, and setup traces for
failing tests
          - 5: Print execution and setup traces for all tests

Fork config:
      --compute-units-per-second <CUPS>
          Sets the number of assumed available compute units per second for
this provider

          default value: 330

          See also --fork-url and
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

      --no-rpc-rate-limit
          Disables rate limiting for this node's provider.

          See also --fork-url and
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

          [aliases: no-rate-limit]

Executor environment config:
      --gas-limit <GAS_LIMIT>
          The block gas limit
```

```
      --code-size-limit <CODE_SIZE>
          EIP-170: Contract code size limit in bytes. Useful to increase this
because of tests. By
          default, it is 0x6000 (~25kb)

      --chain <CHAIN>
          The chain name or EIP-155 chain ID

          [aliases: chain-id]

      --gas-price <GAS_PRICE>
          The gas price

      --block-base-fee-per-gas <FEE>
          The base fee in a block

          [aliases: base-fee]

      --tx-origin <ADDRESS>
          The transaction origin

      --block-coinbase <ADDRESS>
          The coinbase of the block

      --block-timestamp <TIMESTAMP>
          The timestamp of the block

      --block-number <BLOCK>
          The block number

      --block-difficulty <DIFFICULTY>
          The block difficulty

      --block-prevrandao <PREVRANDAO>
          The block prevrandao value. NOTE: Before merge this field was
mix_hash

      --block-gas-limit <GAS_LIMIT>
          The block gas limit

      --memory-limit <MEMORY_LIMIT>
          The memory limit per EVM execution in bytes. If this limit is
exceeded, a `MemoryLimitOOG`
          result is thrown.

          The default is 128MiB.

      --disable-block-gas-limit
          Whether to disable the block gas limit checks

          [aliases: no-gas-limit]

      --isolate
          Whether to enable isolation of calls. In isolation mode all top-level
calls are executed
          as a separate transaction in a separate EVM context, enabling more
```

```
precise gas accounting
          and transaction state changes
```

forge doc

Generate documentation for the project

```
$ forge doc --help
Usage: forge doc [OPTIONS]

Options:
      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

  -o, --out <PATH>
          The doc's output path.

          By default, it is the `docs/` in project root.

  -b, --build
          Build the `mdbook` from generated files

  -s, --serve
          Serve the documentation

      --open
          Open the documentation in a browser after serving

      --hostname <HOSTNAME>
          Hostname for serving documentation

  -p, --port <PORT>
          Port for serving documentation

      --deployments [<DEPLOYMENTS>]
          The relative path to the `hardhat-deploy` or `forge-deploy` artifact
directory. Leave
          blank for default

  -i, --include-libraries
          Whether to create docs for external libraries

  -h, --help
          Print help (see a summary with '-h')
```

forge flatten

Flatten a source file and all of its imports into one file

```
$ forge flatten --help
Usage: forge flatten [OPTIONS] <PATH>

Arguments:
  <PATH>
          The path to the contract to flatten

Options:
  -o, --output <PATH>
          The path to output the flattened contract.

          If not specified, the flattened contract will be output to stdout.

  -h, --help
          Print help (see a summary with '-h')

Project options:
      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
          The contracts source directory

  -R, --remappings <REMAPPINGS>
          The project's remappings

      --remappings-env <ENV>
          The project's remappings from the environment

      --cache-path <PATH>
          The path to the compiler cache

      --lib-paths <PATH>
          The path to the library folder

      --hardhat
          Use the Hardhat-style project layout.

          This is the same as using: `--contracts contracts --lib-paths
node_modules`.

          [aliases: hh]

      --config-path <FILE>
          Path to the config file
```

forge fmt

Format Solidity source files

```
$ forge fmt --help
Usage: forge fmt [OPTIONS] [PATH]...

Arguments:
  [PATH]...
          Path to the file, directory or '-' to read from stdin

Options:
      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

      --check
          Run in 'check' mode.

          Exits with 0 if input is formatted correctly. Exits with 1 if
formatting is required.

  -r, --raw
          In 'check' and stdin modes, outputs raw formatted code instead of the
diff

  -h, --help
          Print help (see a summary with '-h')
```

forge geiger

Detects usage of unsafe cheat codes in a project and its dependencies

```
$ forge geiger --help
Usage: forge geiger [OPTIONS] [PATH]...

Arguments:
  [PATH]...
          Paths to files or directories to detect

Options:
      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

      --check
          Run in "check" mode.

          The exit code of the program will be the number of unsafe cheatcodes
found.

      --ignore <PATH>...
          Globs to ignore

      --full
          Print a report of all files, even if no unsafe functions are found

  -h, --help
          Print help (see a summary with '-h')
```

forge generate

Generate scaffold files

```
$ forge generate --help
Usage: forge generate <COMMAND>

Commands:
  test  Scaffolds test file for given contract
  help  Print this message or the help of the given subcommand(s)

Options:
  -h, --help  Print help
```

forge generate test

Scaffolds test file for given contract

```
$ forge generate test --help
Usage: forge generate test --contract-name <CONTRACT_NAME>

Options:
  -c, --contract-name <CONTRACT_NAME>  Contract name for test generation
  -h, --help                           Print help
```

forge generate-fig-spec

Generate Fig autocompletion spec

```
$ forge generate-fig-spec --help
Usage: forge generate-fig-spec

Options:
  -h, --help  Print help
```

forge init

Create a new Forge project

```
$ forge init --help
Usage: forge init [OPTIONS] [PATH]

Arguments:
  [PATH]
          The root directory of the new project

          [default: .]

Options:
  -t, --template <TEMPLATE>
          The template to start from

  -b, --branch <BRANCH>
          Branch argument that can only be used with template option. If not
specified, the default
          branch is used

      --offline
          Do not install dependencies from the network

          [aliases: no-deps]

      --force
          Create the project even if the specified root directory is not empty

      --vscode
          Create a .vscode/settings.json file with Solidity settings, and
generate a remappings.txt
          file

      --shallow
          Perform shallow clones instead of deep ones.

          Improves performance and reduces disk usage, but prevents switching
branches or tags.

      --no-git
          Install without adding the dependency as a submodule

      --no-commit
          Do not create a commit

  -q, --quiet
          Do not print any messages

  -h, --help
          Print help (see a summary with '-h')
```

forge inspect

Get specialized information about a smart contract

```
$ forge inspect --help
Usage: forge inspect [OPTIONS] <CONTRACT> <FIELD>

Arguments:
  <CONTRACT>
          The identifier of the contract to inspect in the form `(<path>:)?
<contractname>`

  <FIELD>
          The contract artifact field to inspect

          [possible values: abi, bytecode, deployedBytecode, assembly,
assemblyOptimized,
          methodIdentifiers, gasEstimates, storageLayout, devdoc, ir,
irOptimized, metadata,
          userdoc, ewasm, errors, events]

Options:
      --pretty
          Pretty print the selected field, if supported

  -h, --help
          Print help (see a summary with '-h')

Cache options:
      --force
          Clear the cache and artifacts folder and recompile

Build options:
      --no-cache
          Disable the cache

Linker options:
      --libraries <LIBRARIES>
          Set pre-linked libraries

          [env: DAPP_LIBRARIES=]

Compiler options:
      --ignored-error-codes <ERROR_CODES>
          Ignore solc warnings by error code

      --deny-warnings
          Warnings will trigger a compiler error

      --no-auto-detect
          Do not auto-detect the `solc` version

      --use <SOLC_VERSION>
          Specify the solc version, or a path to a local solc, to build with.

          Valid values are in the format `x.y.z`, `solc:x.y.z` or
`path/to/solc`.

      --offline
          Do not access the network.
```

```
            Missing solc versions will not be installed.

        --via-ir
            Use the Yul intermediate representation compilation pipeline

        --silent
            Don't print anything on startup

        --ast
            Includes the AST as JSON in the compiler output

        --evm-version <VERSION>
            The target EVM version

        --optimize
            Activate the Solidity optimizer

        --optimizer-runs <RUNS>
            The number of optimizer runs

        --extra-output <SELECTOR>...
            Extra output to include in the contract's artifact.

            Example keys: evm.assembly, ewasm, ir, irOptimized, metadata

            For a full description, see
            https://docs.soliditylang.org/en/v0.8.13/using-the-
compiler.html#input-description

        --extra-output-files <SELECTOR>...
            Extra output to write to separate files.

            Valid values: metadata, ir, irOptimized, ewasm, evm.assembly

Project options:
  -o, --out <PATH>
            The path to the contract artifacts folder

        --revert-strings <REVERT>
            Revert string configuration.

            Possible values are "default", "strip" (remove), "debug" (Solidity-
generated revert
            strings) and "verboseDebug"

        --build-info
            Generate build info files

        --build-info-path <PATH>
            Output path to directory that build info files will be written to

        --root <PATH>
            The project's root path.

            By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
```

```
            The contracts source directory

     -R, --remappings <REMAPPINGS>
            The project's remappings

         --remappings-env <ENV>
            The project's remappings from the environment

         --cache-path <PATH>
            The path to the compiler cache

         --lib-paths <PATH>
            The path to the library folder

         --hardhat
            Use the Hardhat-style project layout.

            This is the same as using: `--contracts contracts --lib-paths
   node_modules`.

            [aliases: hh]

         --config-path <FILE>
            Path to the config file
```

forge install

Install one or multiple dependencies.

```
$ forge install --help
Usage: forge install [OPTIONS] [DEPENDENCIES]...
    forge install [OPTIONS] <github username>/<github project>@<tag>...
    forge install [OPTIONS] <alias>=<github username>/<github project>@<tag>...
    forge install [OPTIONS] <https:// git url>...

Arguments:
  [DEPENDENCIES]...
          The dependencies to install.

          A dependency can be a raw URL, or the path to a GitHub repository.

          Additionally, a ref can be provided by adding @ to the dependency
path.

          A ref can be: - A branch: master - A tag: v1.2.3 - A commit: 8e8128

          Target installation directory can be added via `<alias>=` suffix. The
dependency will
          installed to `lib/<alias>`.

Options:
      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

      --shallow
          Perform shallow clones instead of deep ones.

          Improves performance and reduces disk usage, but prevents switching
branches or tags.

      --no-git
          Install without adding the dependency as a submodule

      --no-commit
          Do not create a commit

  -q, --quiet
          Do not print any messages

  -h, --help
          Print help (see a summary with '-h')
```

forge remappings

Get the automatically inferred remappings for the project

```
$ forge remappings --help
Usage: forge remappings [OPTIONS]

Options:
      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

      --pretty
          Pretty-print the remappings, grouping each of them by context

  -h, --help
          Print help (see a summary with '-h')
```

forge remove

Remove one or multiple dependencies

```
$ forge remove --help
Usage: forge remove [OPTIONS] [DEPENDENCIES]...

Arguments:
  [DEPENDENCIES]...
          The dependencies you want to remove

Options:
      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

  -f, --force
          Override the up-to-date check

  -h, --help
          Print help (see a summary with '-h')
```

forge script

Run a smart contract as a script, building transactions that can be sent onchain

```
$ forge script --help
Usage: forge script [OPTIONS] <PATH> [ARGS]...

Arguments:
  <PATH>
          The contract you want to run. Either the file path or contract name.

          If multiple contracts exist in the same file you must specify the
target contract with
          --target-contract.

  [ARGS]...
          Arguments to pass to the script function

Options:
      --target-contract <CONTRACT_NAME>
          The name of the contract you want to run

          [aliases: tc]

  -s, --sig <SIG>
          The signature of the function you want to call in the contract, or
raw calldata

          [default: run()]

      --priority-gas-price <PRICE>
          Max priority fee per gas for EIP1559 transactions

          [env: ETH_PRIORITY_GAS_PRICE=]

      --legacy
          Use legacy transactions instead of EIP1559 ones.

          This is auto-enabled for common networks without EIP1559.

      --broadcast
          Broadcasts the transactions

      --skip-simulation
          Skips on-chain simulation

  -g, --gas-estimate-multiplier <GAS_ESTIMATE_MULTIPLIER>
          Relative percentage to multiply gas estimates by

          [default: 130]

      --unlocked
          Send via `eth_sendTransaction` using the `--from` argument or
`$ETH_FROM` as sender

      --resume
          Resumes submitting transactions that failed or timed-out previously.

          It DOES NOT simulate the script again and it expects nonces to have
remained the same.
```

```
          Example: If transaction N has a nonce of 22, then the account should
have a nonce of 22,
          otherwise it fails.

      --multi
          If present, --resume or --verify will be assumed to be a multi chain
deployment

      --debug
          Open the script in the debugger.

          Takes precedence over broadcast.

      --slow
          Makes sure a transaction is sent, only after its previous one has
been confirmed and
          succeeded

      --non-interactive
          Disables interactive prompts that might appear when deploying big
contracts.

          For more info on the contract size limit, see EIP-170:
          <https://eips.ethereum.org/EIPS/eip-170>

      --etherscan-api-key <KEY>
          The Etherscan (or equivalent) API key

          [env: ETHERSCAN_API_KEY=]

      --verify
          Verifies all the contracts found in the receipts of a script, if any

      --json
          Output results in JSON format

      --with-gas-price <PRICE>
          Gas price for legacy transactions, or max fee per gas for EIP1559
transactions

          [env: ETH_GAS_PRICE=]

      --skip <SKIP>...
          Skip building files whose names contain the given filter.

          `test` and `script` are aliases for `.t.sol` and `.s.sol`.

  -h, --help
          Print help (see a summary with '-h')

Cache options:
      --force
          Clear the cache and artifacts folder and recompile

Build options:
      --no-cache
          Disable the cache
```

```
Linker options:
      --libraries <LIBRARIES>
          Set pre-linked libraries

          [env: DAPP_LIBRARIES=]

Compiler options:
      --ignored-error-codes <ERROR_CODES>
          Ignore solc warnings by error code

      --deny-warnings
          Warnings will trigger a compiler error

      --no-auto-detect
          Do not auto-detect the `solc` version

      --use <SOLC_VERSION>
          Specify the solc version, or a path to a local solc, to build with.

          Valid values are in the format `x.y.z`, `solc:x.y.z` or
`path/to/solc`.

      --offline
          Do not access the network.

          Missing solc versions will not be installed.

      --via-ir
          Use the Yul intermediate representation compilation pipeline

      --silent
          Don't print anything on startup

      --ast
          Includes the AST as JSON in the compiler output

      --evm-version <VERSION>
          The target EVM version

      --optimize
          Activate the Solidity optimizer

      --optimizer-runs <RUNS>
          The number of optimizer runs

      --extra-output <SELECTOR>...
          Extra output to include in the contract's artifact.

          Example keys: evm.assembly, ewasm, ir, irOptimized, metadata

          For a full description, see
          https://docs.soliditylang.org/en/v0.8.13/using-the-
compiler.html#input-description

      --extra-output-files <SELECTOR>...
          Extra output to write to separate files.

          Valid values: metadata, ir, irOptimized, ewasm, evm.assembly
```

```
Project options:
  -o, --out <PATH>
          The path to the contract artifacts folder

      --revert-strings <REVERT>
          Revert string configuration.

          Possible values are "default", "strip" (remove), "debug" (Solidity-
generated revert
          strings) and "verboseDebug"

      --build-info
          Generate build info files

      --build-info-path <PATH>
          Output path to directory that build info files will be written to

      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
          The contracts source directory

  -R, --remappings <REMAPPINGS>
          The project's remappings

      --remappings-env <ENV>
          The project's remappings from the environment

      --cache-path <PATH>
          The path to the compiler cache

      --lib-paths <PATH>
          The path to the library folder

      --hardhat
          Use the Hardhat-style project layout.

          This is the same as using: `--contracts contracts --lib-paths
node_modules`.

          [aliases: hh]

      --config-path <FILE>
          Path to the config file

Wallet options - raw:
  -a, --froms [<ADDRESSES>...]
          The sender accounts

          [env: ETH_FROM=]

  -i, --interactives <NUM>
          Open an interactive prompt to enter your private key.
```

```
            Takes a value for the number of keys to enter.

            [default: 0]

        --private-keys <RAW_PRIVATE_KEYS>
            Use the provided private keys

        --private-key <RAW_PRIVATE_KEY>
            Use the provided private key

        --mnemonics <MNEMONICS>
            Use the mnemonic phrases of mnemonic files at the specified paths

        --mnemonic-passphrases <PASSPHRASE>
            Use a BIP39 passphrases for the mnemonic

        --mnemonic-derivation-paths <PATH>
            The wallet derivation path.

            Works with both --mnemonic-path and hardware wallets.

        --mnemonic-indexes <INDEXES>
            Use the private key from the given mnemonic index.

            Can be used with --mnemonics, --ledger, --aws and --trezor.

            [default: 0]

    Wallet options - keystore:
        --keystore <PATHS>
            Use the keystore in the given folder or file

            [env: ETH_KEYSTORE=]
            [aliases: keystores]

        --account <ACCOUNT_NAMES>
            Use a keystore from the default keystores folder
    (~/.foundry/keystores) by its filename

            [env: ETH_KEYSTORE_ACCOUNT=]
            [aliases: accounts]

        --password <PASSWORDS>
            The keystore password.

            Used with --keystore.

        --password-file <PATHS>
            The keystore password file path.

            Used with --keystore.

            [env: ETH_PASSWORD=]

    Wallet options - hardware wallet:
      -l, --ledger
            Use a Ledger hardware wallet
```

```
  -t, --trezor
          Use a Trezor hardware wallet

Wallet options - remote:
      --aws
          Use AWS Key Management Service

EVM options:
  -f, --fork-url <URL>
          Fetch state over a remote endpoint instead of starting from an empty
state.

          If you want to fetch state from a specific block number, see --fork-
block-number.

          [aliases: rpc-url]

      --fork-block-number <BLOCK>
          Fetch state from a specific block number over a remote endpoint.

          See --fork-url.

      --fork-retries <RETRIES>
          Number of retries.

          See --fork-url.

      --fork-retry-backoff <BACKOFF>
          Initial retry backoff on encountering errors.

          See --fork-url.

      --no-storage-caching
          Explicitly disables the use of RPC caching.

          All storage slots are read entirely from the endpoint.

          This flag overrides the project's configuration file.

          See --fork-url.

      --initial-balance <BALANCE>
          The initial balance of deployed test contracts

      --sender <ADDRESS>
          The address which will be executing tests

      --ffi
          Enable the FFI cheatcode

      --always-use-create-2-factory
          Use the create 2 factory in all cases including tests and non-
broadcasting scripts

  -v, --verbosity...
          Verbosity of the EVM.
```

```
          Pass multiple times to increase the verbosity (e.g. -v, -vv, -vvv).

          Verbosity levels:
          - 2: Print logs for all tests
          - 3: Print execution traces for failing tests
          - 4: Print execution traces for all tests, and setup traces for
failing tests
          - 5: Print execution and setup traces for all tests

Fork config:
      --compute-units-per-second <CUPS>
          Sets the number of assumed available compute units per second for
this provider

          default value: 330

          See also --fork-url and
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

      --no-rpc-rate-limit
          Disables rate limiting for this node's provider.

          See also --fork-url and
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

          [aliases: no-rate-limit]

Executor environment config:
      --gas-limit <GAS_LIMIT>
          The block gas limit

      --code-size-limit <CODE_SIZE>
          EIP-170: Contract code size limit in bytes. Useful to increase this
because of tests. By
          default, it is 0x6000 (~25kb)

      --chain <CHAIN>
          The chain name or EIP-155 chain ID

          [aliases: chain-id]

      --gas-price <GAS_PRICE>
          The gas price

      --block-base-fee-per-gas <FEE>
          The base fee in a block

          [aliases: base-fee]

      --tx-origin <ADDRESS>
          The transaction origin

      --block-coinbase <ADDRESS>
          The coinbase of the block

      --block-timestamp <TIMESTAMP>
```

```
          The timestamp of the block

      --block-number <BLOCK>
          The block number

      --block-difficulty <DIFFICULTY>
          The block difficulty

      --block-prevrandao <PREVRANDAO>
          The block prevrandao value. NOTE: Before merge this field was
mix_hash

      --block-gas-limit <GAS_LIMIT>
          The block gas limit

      --memory-limit <MEMORY_LIMIT>
          The memory limit per EVM execution in bytes. If this limit is
exceeded, a `MemoryLimitOOG`
          result is thrown.

          The default is 128MiB.

      --disable-block-gas-limit
          Whether to disable the block gas limit checks

          [aliases: no-gas-limit]

      --isolate
          Whether to enable isolation of calls. In isolation mode all top-level
calls are executed
          as a separate transaction in a separate EVM context, enabling more
precise gas accounting
          and transaction state changes

      --retries <RETRIES>
          Number of attempts for retrying verification

          [default: 5]

      --delay <DELAY>
          Optional delay to apply inbetween verification attempts, in seconds

          [default: 5]

Verifier options:
      --verifier <VERIFIER>
          The contract verification provider to use

          [default: etherscan]
          [possible values: etherscan, sourcify, blockscout]

      --verifier-url <VERIFIER_URL>
          The verifier URL, if using a custom provider

          [env: VERIFIER_URL=]
```

forge selectors

Function selector utilities

```
$ forge selectors --help
Usage: forge selectors <COMMAND>

Commands:
  collision  Check for selector collisions between contracts [aliases: co]
  upload     Upload selectors to registry [aliases: up]
  list       List selectors from current workspace [aliases: ls]
  help       Print this message or the help of the given subcommand(s)

Options:
  -h, --help  Print help
```

forge selectors collision

Check for selector collisions between contracts

```
$ forge selectors collision --help
Usage: forge selectors collision [OPTIONS] <FIRST_CONTRACT> <SECOND_CONTRACT>

Arguments:
  <FIRST_CONTRACT>
          The first of the two contracts for which to look selector collisions
for, in the form
          `(<path>:)?<contractname>`

  <SECOND_CONTRACT>
          The second of the two contracts for which to look selector collisions
for, in the form
          `(<path>:)?<contractname>`

Options:
  -h, --help
          Print help (see a summary with '-h')

Cache options:
      --force
          Clear the cache and artifacts folder and recompile

Build options:
      --no-cache
          Disable the cache

Linker options:
      --libraries <LIBRARIES>
          Set pre-linked libraries

          [env: DAPP_LIBRARIES=]

Compiler options:
      --ignored-error-codes <ERROR_CODES>
          Ignore solc warnings by error code

      --deny-warnings
          Warnings will trigger a compiler error

      --no-auto-detect
          Do not auto-detect the `solc` version

      --use <SOLC_VERSION>
          Specify the solc version, or a path to a local solc, to build with.

          Valid values are in the format `x.y.z`, `solc:x.y.z` or
`path/to/solc`.

      --offline
          Do not access the network.

          Missing solc versions will not be installed.

      --via-ir
          Use the Yul intermediate representation compilation pipeline

      --silent
```

```
            Don't print anything on startup

        --ast
            Includes the AST as JSON in the compiler output

        --evm-version <VERSION>
            The target EVM version

        --optimize
            Activate the Solidity optimizer

        --optimizer-runs <RUNS>
            The number of optimizer runs

        --extra-output <SELECTOR>...
            Extra output to include in the contract's artifact.

            Example keys: evm.assembly, ewasm, ir, irOptimized, metadata

            For a full description, see
            https://docs.soliditylang.org/en/v0.8.13/using-the-
compiler.html#input-description

        --extra-output-files <SELECTOR>...
            Extra output to write to separate files.

            Valid values: metadata, ir, irOptimized, ewasm, evm.assembly

Project options:
  -o, --out <PATH>
            The path to the contract artifacts folder

        --revert-strings <REVERT>
            Revert string configuration.

            Possible values are "default", "strip" (remove), "debug" (Solidity-
generated revert
            strings) and "verboseDebug"

        --build-info
            Generate build info files

        --build-info-path <PATH>
            Output path to directory that build info files will be written to

        --root <PATH>
            The project's root path.

            By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
            The contracts source directory

  -R, --remappings <REMAPPINGS>
            The project's remappings

        --remappings-env <ENV>
```

```
            The project's remappings from the environment

        --cache-path <PATH>
            The path to the compiler cache

        --lib-paths <PATH>
            The path to the library folder

        --hardhat
            Use the Hardhat-style project layout.

            This is the same as using: `--contracts contracts --lib-paths
    node_modules`.

            [aliases: hh]

        --config-path <FILE>
            Path to the config file
```

forge selectors upload

Upload selectors to registry

```
$ forge selectors upload --help
Usage: forge selectors upload [OPTIONS] [CONTRACT]

Arguments:
  [CONTRACT]
          The name of the contract to upload selectors for

Options:
      --all
          Upload selectors for all contracts in the project

  -h, --help
          Print help (see a summary with '-h')

Project options:
      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
          The contracts source directory

  -R, --remappings <REMAPPINGS>
          The project's remappings

      --remappings-env <ENV>
          The project's remappings from the environment

      --cache-path <PATH>
          The path to the compiler cache

      --lib-paths <PATH>
          The path to the library folder

      --hardhat
          Use the Hardhat-style project layout.

          This is the same as using: `--contracts contracts --lib-paths
node_modules`.

          [aliases: hh]

      --config-path <FILE>
          Path to the config file
```

forge selectors list

List selectors from current workspace

```
$ forge selectors list --help
Usage: forge selectors list [OPTIONS] [CONTRACT]

Arguments:
  [CONTRACT]
          The name of the contract to list selectors for.

Options:
  -h, --help
          Print help (see a summary with '-h')

Project options:
      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
          The contracts source directory

  -R, --remappings <REMAPPINGS>
          The project's remappings

      --remappings-env <ENV>
          The project's remappings from the environment

      --cache-path <PATH>
          The path to the compiler cache

      --lib-paths <PATH>
          The path to the library folder

      --hardhat
          Use the Hardhat-style project layout.

          This is the same as using: `--contracts contracts --lib-paths
node_modules`.

          [aliases: hh]

      --config-path <FILE>
          Path to the config file
```

forge snapshot

Create a snapshot of each test's gas usage

```
$ forge snapshot --help
Usage: forge snapshot [OPTIONS]

Options:
      --diff [<SNAPSHOT_FILE>]
          Output a diff against a pre-existing snapshot.

          By default, the comparison is done with .gas-snapshot.

      --check [<SNAPSHOT_FILE>]
          Compare against a pre-existing snapshot, exiting with code 1 if they
do not match.

          Outputs a diff if the snapshots do not match.

          By default, the comparison is done with .gas-snapshot.

      --snap <FILE>
          Output file for the snapshot

          [default: .gas-snapshot]

      --tolerance <SNAPSHOT_THRESHOLD>
          Tolerates gas deviations up to the specified percentage

  -h, --help
          Print help (see a summary with '-h')

Test options:
      --debug <TEST_FUNCTION>
          Run a test in the debugger.

          The argument passed to this flag is the name of the test function you
want to run, and it
          works the same as --match-test.

          If more than one test matches your specified criteria, you must add
additional filters
          until only one test is found (see --match-contract and --match-path).

          The matching test will be opened in the debugger regardless of the
outcome of the test.

          If the matching test is a fuzz test, then it will open the debugger
on the first failure
          case. If the fuzz test does not fail, it will open the debugger on
the last fuzz case.

          For more fine-grained control of which fuzz case is run, see forge
run.

      --gas-report
          Print a gas report

          [env: FORGE_GAS_REPORT=]

      --allow-failure
```

```
            Exit with code 0 even if a test fails

            [env: FORGE_ALLOW_FAILURE=]

        --fail-fast
            Stop running tests after the first failure

        --etherscan-api-key <KEY>
            The Etherscan (or equivalent) API key

            [env: ETHERSCAN_API_KEY=]

        --fuzz-seed <FUZZ_SEED>
            Set seed used to generate randomness during your fuzz runs

        --fuzz-runs <RUNS>
            [env: FOUNDRY_FUZZ_RUNS=]

        --fuzz-input-file <FUZZ_INPUT_FILE>
            File to rerun fuzz failures from

    Display options:
      -j, --json
            Output test results in JSON format

      -l, --list
            List tests instead of running them

        --summary
            Print test summary table

        --detailed
            Print detailed test summary table

    Test filtering:
        --match-test <REGEX>
            Only run test functions matching the specified regex pattern

            [aliases: mt]

        --no-match-test <REGEX>
            Only run test functions that do not match the specified regex pattern

            [aliases: nmt]

        --match-contract <REGEX>
            Only run tests in contracts matching the specified regex pattern

            [aliases: mc]

        --no-match-contract <REGEX>
            Only run tests in contracts that do not match the specified regex
    pattern

            [aliases: nmc]

        --match-path <GLOB>
            Only run tests in source files matching the specified glob pattern
```

```
          [aliases: mp]

      --no-match-path <GLOB>
          Only run tests in source files that do not match the specified glob
pattern

          [aliases: nmp]

EVM options:
  -f, --fork-url <URL>
          Fetch state over a remote endpoint instead of starting from an empty
state.

          If you want to fetch state from a specific block number, see --fork-
block-number.

          [aliases: rpc-url]

      --fork-block-number <BLOCK>
          Fetch state from a specific block number over a remote endpoint.

          See --fork-url.

      --fork-retries <RETRIES>
          Number of retries.

          See --fork-url.

      --fork-retry-backoff <BACKOFF>
          Initial retry backoff on encountering errors.

          See --fork-url.

      --no-storage-caching
          Explicitly disables the use of RPC caching.

          All storage slots are read entirely from the endpoint.

          This flag overrides the project's configuration file.

          See --fork-url.

      --initial-balance <BALANCE>
          The initial balance of deployed test contracts

      --sender <ADDRESS>
          The address which will be executing tests

      --ffi
          Enable the FFI cheatcode

      --always-use-create-2-factory
          Use the create 2 factory in all cases including tests and non-
broadcasting scripts

  -v, --verbosity...
          Verbosity of the EVM.
```

```
          Pass multiple times to increase the verbosity (e.g. -v, -vv, -vvv).

          Verbosity levels:
          - 2: Print logs for all tests
          - 3: Print execution traces for failing tests
          - 4: Print execution traces for all tests, and setup traces for
failing tests
          - 5: Print execution and setup traces for all tests

Fork config:
      --compute-units-per-second <CUPS>
          Sets the number of assumed available compute units per second for
this provider

          default value: 330

          See also --fork-url and
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

      --no-rpc-rate-limit
          Disables rate limiting for this node's provider.

          See also --fork-url and
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

          [aliases: no-rate-limit]

Executor environment config:
      --gas-limit <GAS_LIMIT>
          The block gas limit

      --code-size-limit <CODE_SIZE>
          EIP-170: Contract code size limit in bytes. Useful to increase this
because of tests. By
          default, it is 0x6000 (~25kb)

      --chain <CHAIN>
          The chain name or EIP-155 chain ID

          [aliases: chain-id]

      --gas-price <GAS_PRICE>
          The gas price

      --block-base-fee-per-gas <FEE>
          The base fee in a block

          [aliases: base-fee]

      --tx-origin <ADDRESS>
          The transaction origin

      --block-coinbase <ADDRESS>
          The coinbase of the block
```

```
      --block-timestamp <TIMESTAMP>
          The timestamp of the block

      --block-number <BLOCK>
          The block number

      --block-difficulty <DIFFICULTY>
          The block difficulty

      --block-prevrandao <PREVRANDAO>
          The block prevrandao value. NOTE: Before merge this field was
mix_hash

      --block-gas-limit <GAS_LIMIT>
          The block gas limit

      --memory-limit <MEMORY_LIMIT>
          The memory limit per EVM execution in bytes. If this limit is
exceeded, a `MemoryLimitOOG`
          result is thrown.

          The default is 128MiB.

      --disable-block-gas-limit
          Whether to disable the block gas limit checks

          [aliases: no-gas-limit]

      --isolate
          Whether to enable isolation of calls. In isolation mode all top-level
calls are executed
          as a separate transaction in a separate EVM context, enabling more
precise gas accounting
          and transaction state changes

Cache options:
      --force
          Clear the cache and artifacts folder and recompile

Build options:
      --no-cache
          Disable the cache

Linker options:
      --libraries <LIBRARIES>
          Set pre-linked libraries

          [env: DAPP_LIBRARIES=]

Compiler options:
      --ignored-error-codes <ERROR_CODES>
          Ignore solc warnings by error code

      --deny-warnings
          Warnings will trigger a compiler error

      --no-auto-detect
          Do not auto-detect the `solc` version
```

```
      --use <SOLC_VERSION>
          Specify the solc version, or a path to a local solc, to build with.

          Valid values are in the format `x.y.z`, `solc:x.y.z` or
`path/to/solc`.

      --offline
          Do not access the network.

          Missing solc versions will not be installed.

      --via-ir
          Use the Yul intermediate representation compilation pipeline

      --silent
          Don't print anything on startup

      --ast
          Includes the AST as JSON in the compiler output

      --evm-version <VERSION>
          The target EVM version

      --optimize
          Activate the Solidity optimizer

      --optimizer-runs <RUNS>
          The number of optimizer runs

      --extra-output <SELECTOR>...
          Extra output to include in the contract's artifact.

          Example keys: evm.assembly, ewasm, ir, irOptimized, metadata

          For a full description, see
          https://docs.soliditylang.org/en/v0.8.13/using-the-
compiler.html#input-description

      --extra-output-files <SELECTOR>...
          Extra output to write to separate files.

          Valid values: metadata, ir, irOptimized, ewasm, evm.assembly

Project options:
  -o, --out <PATH>
          The path to the contract artifacts folder

      --revert-strings <REVERT>
          Revert string configuration.

          Possible values are "default", "strip" (remove), "debug" (Solidity-
generated revert
          strings) and "verboseDebug"

      --build-info
          Generate build info files
```

```
      --build-info-path <PATH>
          Output path to directory that build info files will be written to

      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
          The contracts source directory

  -R, --remappings <REMAPPINGS>
          The project's remappings

      --remappings-env <ENV>
          The project's remappings from the environment

      --cache-path <PATH>
          The path to the compiler cache

      --lib-paths <PATH>
          The path to the library folder

      --hardhat
          Use the Hardhat-style project layout.

          This is the same as using: `--contracts contracts --lib-paths
node_modules`.

          [aliases: hh]

      --config-path <FILE>
          Path to the config file

Watch options:
  -w, --watch [<PATH>...]
          Watch the given files or directories for changes.

          If no paths are provided, the source and test directories of the
project are watched.

      --no-restart
          Do not restart the command while it's still running

      --run-all
          Explicitly re-run all tests when a change is made.

          By default, only the tests of the last modified test file are
executed.

      --watch-delay <DELAY>
          File update debounce delay.

          During the delay, incoming change events are accumulated and only
once the delay has
          passed, is an action taken. Note that this does not mean a command
will be started: if
```

```
          --no-restart is given and a command is already running, the outcome
of the action will be
          to do nothing.

          Defaults to 50ms. Parses as decimal seconds by default, but using an
integer with the `ms`
          suffix may be more convenient.

          When using --poll mode, you'll want a larger duration, or risk
overloading disk I/O.

      --asc
          Sort results by gas used (ascending)

      --desc
          Sort results by gas used (descending)

      --min <MIN_GAS>
          Only include tests that used more gas that the given amount

      --max <MAX_GAS>
          Only include tests that used less gas that the given amount
```

forge test

Run the project's tests

```
$ forge test --help
Usage: forge test [OPTIONS]

Options:
  -h, --help
          Print help (see a summary with '-h')

Test options:
      --debug <TEST_FUNCTION>
          Run a test in the debugger.

          The argument passed to this flag is the name of the test function you
want to run, and it
          works the same as --match-test.

          If more than one test matches your specified criteria, you must add
additional filters
          until only one test is found (see --match-contract and --match-path).

          The matching test will be opened in the debugger regardless of the
outcome of the test.

          If the matching test is a fuzz test, then it will open the debugger
on the first failure
          case. If the fuzz test does not fail, it will open the debugger on
the last fuzz case.

          For more fine-grained control of which fuzz case is run, see forge
run.

      --gas-report
          Print a gas report

          [env: FORGE_GAS_REPORT=]

      --allow-failure
          Exit with code 0 even if a test fails

          [env: FORGE_ALLOW_FAILURE=]

      --fail-fast
          Stop running tests after the first failure

      --etherscan-api-key <KEY>
          The Etherscan (or equivalent) API key

          [env: ETHERSCAN_API_KEY=]

      --fuzz-seed <FUZZ_SEED>
          Set seed used to generate randomness during your fuzz runs

      --fuzz-runs <RUNS>
          [env: FOUNDRY_FUZZ_RUNS=]

      --fuzz-input-file <FUZZ_INPUT_FILE>
          File to rerun fuzz failures from
```

```
Display options:
  -j, --json
          Output test results in JSON format

  -l, --list
          List tests instead of running them

      --summary
          Print test summary table

      --detailed
          Print detailed test summary table

Test filtering:
      --match-test <REGEX>
          Only run test functions matching the specified regex pattern

          [aliases: mt]

      --no-match-test <REGEX>
          Only run test functions that do not match the specified regex pattern

          [aliases: nmt]

      --match-contract <REGEX>
          Only run tests in contracts matching the specified regex pattern

          [aliases: mc]

      --no-match-contract <REGEX>
          Only run tests in contracts that do not match the specified regex
pattern

          [aliases: nmc]

      --match-path <GLOB>
          Only run tests in source files matching the specified glob pattern

          [aliases: mp]

      --no-match-path <GLOB>
          Only run tests in source files that do not match the specified glob
pattern

          [aliases: nmp]

EVM options:
  -f, --fork-url <URL>
          Fetch state over a remote endpoint instead of starting from an empty
state.

          If you want to fetch state from a specific block number, see --fork-
block-number.

          [aliases: rpc-url]

      --fork-block-number <BLOCK>
          Fetch state from a specific block number over a remote endpoint.
```

```
          See --fork-url.

      --fork-retries <RETRIES>
          Number of retries.

          See --fork-url.

      --fork-retry-backoff <BACKOFF>
          Initial retry backoff on encountering errors.

          See --fork-url.

      --no-storage-caching
          Explicitly disables the use of RPC caching.

          All storage slots are read entirely from the endpoint.

          This flag overrides the project's configuration file.

          See --fork-url.

      --initial-balance <BALANCE>
          The initial balance of deployed test contracts

      --sender <ADDRESS>
          The address which will be executing tests

      --ffi
          Enable the FFI cheatcode

      --always-use-create-2-factory
          Use the create 2 factory in all cases including tests and non-
broadcasting scripts

  -v, --verbosity...
          Verbosity of the EVM.

          Pass multiple times to increase the verbosity (e.g. -v, -vv, -vvv).

          Verbosity levels:
          - 2: Print logs for all tests
          - 3: Print execution traces for failing tests
          - 4: Print execution traces for all tests, and setup traces for
failing tests
          - 5: Print execution and setup traces for all tests

Fork config:
      --compute-units-per-second <CUPS>
          Sets the number of assumed available compute units per second for
this provider

          default value: 330

          See also --fork-url and
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second
```

```
      --no-rpc-rate-limit
          Disables rate limiting for this node's provider.

          See also --fork-url and
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

          [aliases: no-rate-limit]

Executor environment config:
      --gas-limit <GAS_LIMIT>
          The block gas limit

      --code-size-limit <CODE_SIZE>
          EIP-170: Contract code size limit in bytes. Useful to increase this
because of tests. By
          default, it is 0x6000 (~25kb)

      --chain <CHAIN>
          The chain name or EIP-155 chain ID

          [aliases: chain-id]

      --gas-price <GAS_PRICE>
          The gas price

      --block-base-fee-per-gas <FEE>
          The base fee in a block

          [aliases: base-fee]

      --tx-origin <ADDRESS>
          The transaction origin

      --block-coinbase <ADDRESS>
          The coinbase of the block

      --block-timestamp <TIMESTAMP>
          The timestamp of the block

      --block-number <BLOCK>
          The block number

      --block-difficulty <DIFFICULTY>
          The block difficulty

      --block-prevrandao <PREVRANDAO>
          The block prevrandao value. NOTE: Before merge this field was
mix_hash

      --block-gas-limit <GAS_LIMIT>
          The block gas limit

      --memory-limit <MEMORY_LIMIT>
          The memory limit per EVM execution in bytes. If this limit is
exceeded, a `MemoryLimitOOG`
          result is thrown.
```

```
            The default is 128MiB.

        --disable-block-gas-limit
            Whether to disable the block gas limit checks

            [aliases: no-gas-limit]

        --isolate
            Whether to enable isolation of calls. In isolation mode all top-level
calls are executed
            as a separate transaction in a separate EVM context, enabling more
precise gas accounting
            and transaction state changes

Cache options:
        --force
            Clear the cache and artifacts folder and recompile

Build options:
        --no-cache
            Disable the cache

Linker options:
        --libraries <LIBRARIES>
            Set pre-linked libraries

            [env: DAPP_LIBRARIES=]

Compiler options:
        --ignored-error-codes <ERROR_CODES>
            Ignore solc warnings by error code

        --deny-warnings
            Warnings will trigger a compiler error

        --no-auto-detect
            Do not auto-detect the `solc` version

        --use <SOLC_VERSION>
            Specify the solc version, or a path to a local solc, to build with.

            Valid values are in the format `x.y.z`, `solc:x.y.z` or
`path/to/solc`.

        --offline
            Do not access the network.

            Missing solc versions will not be installed.

        --via-ir
            Use the Yul intermediate representation compilation pipeline

        --silent
            Don't print anything on startup

        --ast
            Includes the AST as JSON in the compiler output
```

```
      --evm-version <VERSION>
          The target EVM version

      --optimize
          Activate the Solidity optimizer

      --optimizer-runs <RUNS>
          The number of optimizer runs

      --extra-output <SELECTOR>...
          Extra output to include in the contract's artifact.

          Example keys: evm.assembly, ewasm, ir, irOptimized, metadata

          For a full description, see
          https://docs.soliditylang.org/en/v0.8.13/using-the-
compiler.html#input-description

      --extra-output-files <SELECTOR>...
          Extra output to write to separate files.

          Valid values: metadata, ir, irOptimized, ewasm, evm.assembly

Project options:
  -o, --out <PATH>
          The path to the contract artifacts folder

      --revert-strings <REVERT>
          Revert string configuration.

          Possible values are "default", "strip" (remove), "debug" (Solidity-
generated revert
          strings) and "verboseDebug"

      --build-info
          Generate build info files

      --build-info-path <PATH>
          Output path to directory that build info files will be written to

      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
          The contracts source directory

  -R, --remappings <REMAPPINGS>
          The project's remappings

      --remappings-env <ENV>
          The project's remappings from the environment

      --cache-path <PATH>
          The path to the compiler cache
```

```
      --lib-paths <PATH>
          The path to the library folder

      --hardhat
          Use the Hardhat-style project layout.

          This is the same as using: `--contracts contracts --lib-paths
node_modules`.

          [aliases: hh]

      --config-path <FILE>
          Path to the config file

Watch options:
  -w, --watch [<PATH>...]
          Watch the given files or directories for changes.

          If no paths are provided, the source and test directories of the
project are watched.

      --no-restart
          Do not restart the command while it's still running

      --run-all
          Explicitly re-run all tests when a change is made.

          By default, only the tests of the last modified test file are
executed.

      --watch-delay <DELAY>
          File update debounce delay.

          During the delay, incoming change events are accumulated and only
once the delay has
          passed, is an action taken. Note that this does not mean a command
will be started: if
          --no-restart is given and a command is already running, the outcome
of the action will be
          to do nothing.

          Defaults to 50ms. Parses as decimal seconds by default, but using an
integer with the `ms`
          suffix may be more convenient.

          When using --poll mode, you'll want a larger duration, or risk
overloading disk I/O.
```

forge tree

Display a tree visualization of the project's dependency graph

```
$ forge tree --help
Usage: forge tree [OPTIONS]

Options:
      --no-dedupe
          Do not de-duplicate (repeats all shared dependencies)

      --charset <CHARSET>
          Character set to use in output.

          [possible values: utf8, ascii]

          [default: utf8]

  -h, --help
          Print help (see a summary with '-h')

Project options:
      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
          The contracts source directory

  -R, --remappings <REMAPPINGS>
          The project's remappings

      --remappings-env <ENV>
          The project's remappings from the environment

      --cache-path <PATH>
          The path to the compiler cache

      --lib-paths <PATH>
          The path to the library folder

      --hardhat
          Use the Hardhat-style project layout.

          This is the same as using: `--contracts contracts --lib-paths
node_modules`.

          [aliases: hh]

      --config-path <FILE>
          Path to the config file
```

forge update

Update one or multiple dependencies.

```
$ forge update --help
Usage: forge update [OPTIONS] [DEPENDENCIES]...

Arguments:
  [DEPENDENCIES]...
          The dependencies you want to update

Options:
      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

  -f, --force
          Override the up-to-date check

  -r, --recursive
          Recursively update submodules

  -h, --help
          Print help (see a summary with '-h')
```

forge verify-check

Check verification status on Etherscan

```
$ forge verify-check --help
Usage: forge verify-check [OPTIONS] <ID>

Arguments:
  <ID>
          The verification ID.

          For Etherscan - Submission GUID.

          For Sourcify - Contract Address.

Options:
      --retries <RETRIES>
          Number of attempts for retrying verification

          [default: 5]

      --delay <DELAY>
          Optional delay to apply inbetween verification attempts, in seconds

          [default: 5]

  -e, --etherscan-api-key <KEY>
          The Etherscan (or equivalent) API key

          [env: ETHERSCAN_API_KEY=]

  -c, --chain <CHAIN>
          The chain name or EIP-155 chain ID

          [env: CHAIN=]

  -h, --help
          Print help (see a summary with '-h')

Verifier options:
      --verifier <VERIFIER>
          The contract verification provider to use

          [default: etherscan]
          [possible values: etherscan, sourcify, blockscout]

      --verifier-url <VERIFIER_URL>
          The verifier URL, if using a custom provider

          [env: VERIFIER_URL=]
```

forge verify-contract

Verify smart contracts on Etherscan

```
$ forge verify-contract --help
Usage: forge verify-contract [OPTIONS] <ADDRESS> <CONTRACT>

Arguments:
  <ADDRESS>
          The address of the contract to verify

  <CONTRACT>
          The contract identifier in the form `<path>:<contractname>`

Options:
      --constructor-args <ARGS>
          The ABI-encoded constructor arguments

          [aliases: encoded-constructor-args]

      --constructor-args-path <PATH>
          The path to a file containing the constructor arguments

      --guess-constructor-args
          Try to extract constructor arguments from on-chain creation code

      --compiler-version <VERSION>
          The `solc` version to use to build the smart contract

      --num-of-optimizations <NUM>
          The number of optimization runs used to build the smart contract

          [aliases: optimizer-runs]

      --flatten
          Flatten the source code before verifying

  -f, --force
          Do not compile the flattened smart contract before verifying (if --
flatten is passed)

      --skip-is-verified-check
          Do not check if the contract is already verified before verifying

      --watch
          Wait for verification result after submission

      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

      --show-standard-json-input
          Prints the standard json compiler input.

          The standard json compiler input can be used to manually submit
contract verification in
          the browser.

      --via-ir
```

```
            Use the Yul intermediate representation compilation pipeline

        --evm-version <EVM_VERSION>
            The EVM version to use.

            Overrides the version specified in the config.

    -e, --etherscan-api-key <KEY>
            The Etherscan (or equivalent) API key

            [env: ETHERSCAN_API_KEY=]

    -c, --chain <CHAIN>
            The chain name or EIP-155 chain ID

            [env: CHAIN=]

    -r, --rpc-url <URL>
            The RPC endpoint

            [env: ETH_RPC_URL=]

        --flashbots
            Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
            transaction privately with all registered builders.
            https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

        --jwt-secret <JWT_SECRET>
            JWT Secret for the RPC endpoint.

            The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
            used to simulate a CL `engine_forkchoiceUpdated` call:

            cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
            "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

            [env: ETH_RPC_JWT_SECRET=]

        --retries <RETRIES>
            Number of attempts for retrying verification

            [default: 5]

        --delay <DELAY>
            Optional delay to apply inbetween verification attempts, in seconds

            [default: 5]

    -h, --help
            Print help (see a summary with '-h')
```

```
Linker options:
      --libraries <LIBRARIES>
          Set pre-linked libraries

          [env: DAPP_LIBRARIES=]

Verifier options:
      --verifier <VERIFIER>
          The contract verification provider to use

          [default: etherscan]
          [possible values: etherscan, sourcify, blockscout]

      --verifier-url <VERIFIER_URL>
          The verifier URL, if using a custom provider

          [env: VERIFIER_URL=]
```

cast

Perform Ethereum RPC calls from the comfort of your command line

```
$ cast --help
Usage: cast <COMMAND>

Commands:
  4byte                  Get the function signatures for the given selector
from
                             https://openchain.xyz [aliases: 4, 4b]
  4byte-decode           Decode ABI-encoded calldata using
https://openchain.xyz [aliases: 4d,
                             4bd]
  4byte-event            Get the event signature for a given topic 0 from
https://openchain.xyz
                             [aliases: 4e, 4be, topic0-event, t0e]
  abi-decode             Decode ABI-encoded input or output data [aliases: ad,
--abi-decode]
  abi-encode             ABI encode the given function argument, excluding the
selector
                             [aliases: ae]
  access-list            Create an access list for a transaction [aliases: ac,
acl]
  address-zero           Prints the zero address [aliases: --address-zero, az]
  admin                  Fetch the EIP-1967 admin account [aliases: adm]
  age                    Get the timestamp of a block [aliases: a]
  balance                Get the balance of an account in wei [aliases: b]
  base-fee               Get the basefee of a block [aliases: ba, fee, basefee]
  bind                   Generate a rust binding from a given ABI [aliases: bi]
  block                  Get information about a block [aliases: bl]
  block-number           Get the latest block number [aliases: bn]
  call                   Perform a call on an account without publishing a
transaction [aliases:
                             c]
  calldata               ABI-encode a function with arguments [aliases: cd]
  calldata-decode        Decode ABI-encoded input data [aliases: --calldata-
decode, cdd]
  chain                  Get the symbolic name of the current chain
  chain-id               Get the Ethereum chain ID [aliases: ci, cid]
  client                 Get the current client version [aliases: cl]
  code                   Get the runtime bytecode of a contract [aliases: co]
  codesize               Get the runtime bytecode size of a contract [aliases:
cs]
  completions            Generate shell completions script [aliases: com]
  compute-address        Compute the contract address from a given nonce and
deployer address
                             [aliases: ca]
  concat-hex             Concatenate hex strings [aliases: --concat-hex, ch]
  create2                Generate a deterministic contract address using
CREATE2 [aliases: c2]
  decode-transaction     Decodes a raw signed EIP 2718 typed transaction
[aliases: dt]
  disassemble            Disassembles hex encoded bytecode into individual /
human readable
                             opcodes [aliases: da]
  estimate               Estimate the gas cost of a transaction [aliases: e]
  etherscan-source       Get the source code of a contract from Etherscan
[aliases: et, src]
  find-block             Get the block number closest to the provided timestamp
[aliases: f]
```

```
  format-bytes32-string  Formats a string into bytes32 encoding [aliases:
                             --format-bytes32-string]
  from-bin               Convert binary data into hex data [aliases: --from-
bin, from-binx, fb]
  from-fixed-point       Convert a fixed point number into an integer [aliases:
--from-fix, ff]
  from-rlp               Decodes RLP encoded data [aliases: --from-rlp]
  from-utf8              Convert UTF8 text to hex [aliases: --from-ascii, --
from-utf8,
                             from-ascii, fu, fa]
  from-wei               Convert wei into an ETH amount [aliases: --from-wei,
fw]
  gas-price              Get the current gas price [aliases: g]
  generate-fig-spec      Generate Fig autocompletion spec [aliases: fig]
  hash-zero              Prints the zero hash [aliases: --hash-zero, hz]
  help                   Print this message or the help of the given
subcommand(s)
  implementation         Fetch the EIP-1967 implementation account [aliases:
impl]
  index                  Compute the storage slot for an entry in a mapping
[aliases: in]
  interface              Generate a Solidity interface from a given ABI
[aliases: i]
  keccak                 Hash arbitrary data using Keccak-256 [aliases: k]
  logs                   Get logs by signature or topic [aliases: l]
  lookup-address         Perform an ENS reverse lookup [aliases: la]
  max-int                Prints the maximum value of the given integer type
[aliases: --max-int,
                             maxi]
  max-uint               Prints the maximum value of the given integer type
[aliases:
                             --max-uint, maxu]
  min-int                Prints the minimum value of the given integer type
[aliases: --min-int,
                             mini]
  mktx                   Build and sign a transaction [aliases: m]
  namehash               Calculate the ENS namehash of a name [aliases: na, nh]
  nonce                  Get the nonce for an account [aliases: n]
  parse-bytes32-address  Parses a checksummed address from bytes32 encoding.
[aliases:
                             --parse-bytes32-address]
  parse-bytes32-string   Parses a string from bytes32 encoding [aliases: --
parse-bytes32-string]
  pretty-calldata        Pretty print calldata [aliases: pc]
  proof                  Generate a storage proof for a given storage slot
[aliases: pr]
  publish                Publish a raw transaction to the network [aliases: p]
  receipt                Get the transaction receipt for a transaction
[aliases: re]
  resolve-name           Perform an ENS lookup [aliases: rn]
  rpc                    Perform a raw JSON-RPC request [aliases: rp]
  run                    Runs a published transaction in a local environment
and prints the
                             trace [aliases: r]
  selectors              Extracts function selectors and arguments from
bytecode [aliases: sel]
  send                   Sign and publish a transaction [aliases: s]
  shl                    Perform a left shifting operation
```

```
  shr                     Perform a right shifting operation
  sig                     Get the selector for a function [aliases: si]
  sig-event               Generate event signatures from event string [aliases:
se]
  storage                 Get the raw value of a contract's storage slot
[aliases: st]
  to-ascii                Convert hex data to an ASCII string [aliases: --to-
ascii, tas, 2as]
  to-base                 Converts a number of one base to another [aliases: --
to-base,
                              --to-radix, to-radix, tr, 2r]
  to-bytes32              Right-pads hex data to 32 bytes [aliases: --to-
bytes32, tb, 2b]
  to-check-sum-address    Convert an address to a checksummed format (EIP-55)
[aliases:
                              --to-checksum-address, --to-checksum, to-checksum,
ta, 2a]
  to-dec                  Converts a number of one base to decimal [aliases: --
to-dec, td, 2d]
  to-fixed-point          Convert an integer into a fixed point number [aliases:
--to-fix, tf,
                              2f]
  to-hex                  Converts a number of one base to another [aliases: --
to-hex, th, 2h]
  to-hexdata              Normalize the input to lowercase, 0x-prefixed hex
[aliases:
                              --to-hexdata, thd, 2hd]
  to-int256               Convert a number to a hex-encoded int256 [aliases: --
to-int256, ti, 2i]
  to-rlp                  RLP encodes hex data, or an array of hex data
[aliases: --to-rlp]
  to-uint256              Convert a number to a hex-encoded uint256 [aliases: --
to-uint256, tu,
                              2u]
  to-unit                 Convert an ETH amount into another unit (ether, gwei
or wei) [aliases:
                              --to-unit, tun, 2un]
  to-wei                  Convert an ETH amount to wei [aliases: --to-wei, tw,
2w]
  tx                      Get information about a transaction [aliases: t]
  upload-signature        Upload the given signatures to https://openchain.xyz
[aliases: ups]
  wallet                  Wallet management utilities [aliases: w]

Options:
  -h, --help     Print help
  -V, --version  Print version

Find more information in the book:
http://book.getfoundry.sh/reference/cast/cast.html
```

cast 4byte

Get the function signatures for the given selector from https://openchain.xyz

```
$ cast 4byte --help
Usage: cast 4byte [SELECTOR]

Arguments:
  [SELECTOR]  The function selector

Options:
  -h, --help  Print help
```

cast 4byte-decode

Decode ABI-encoded calldata using https://openchain.xyz

```
$ cast 4byte-decode --help
Usage: cast 4byte-decode [CALLDATA]

Arguments:
  [CALLDATA]  The ABI-encoded calldata

Options:
  -h, --help  Print help
```

cast 4byte-event

Get the event signature for a given topic 0 from https://openchain.xyz

```
$ cast 4byte-event --help
Usage: cast 4byte-event [TOPIC_0]

Arguments:
  [TOPIC_0]  Topic 0

Options:
  -h, --help  Print help
```

cast abi-decode

Decode ABI-encoded input or output data.

```
$ cast abi-decode --help
Usage: cast abi-decode [OPTIONS] <SIG> <CALLDATA>

Arguments:
  <SIG>
          The function signature in the format `<name>(<in-types>)(<out-
types>)`

  <CALLDATA>
          The ABI-encoded calldata

Options:
  -h, --help
          Print help (see a summary with '-h')

Decode input data instead of output data:
  -i, --input
          Whether to decode the input or output data
```

cast abi-encode

ABI encode the given function argument, excluding the selector

```
$ cast abi-encode --help
Usage: cast abi-encode [OPTIONS] <SIG> [ARGS]...

Arguments:
  <SIG>      The function signature
  [ARGS]...  The arguments of the function

Options:
      --packed  Whether to use packed encoding
  -h, --help    Print help
```

cast access-list

Create an access list for a transaction

```
$ cast access-list --help
Usage: cast access-list [OPTIONS] [TO] [SIG] [ARGS]...

Arguments:
  [TO]
          The destination of the transaction

  [SIG]
          The signature of the function to call

  [ARGS]...
          The arguments of the function to call

Options:
      --data <DATA>
          The data for the transaction

  -B, --block <BLOCK>
          The block height to query at.

          Can also be the tags earliest, finalized, safe, latest, or pending.

  -h, --help
          Print help (see a summary with '-h')

Display options:
  -j, --json
          Print the access list as JSON

Transaction options:
      --gas-limit <GAS_LIMIT>
          Gas limit for the transaction

          [env: ETH_GAS_LIMIT=]

      --gas-price <PRICE>
          Gas price for legacy transactions, or max fee per gas for EIP1559
transactions

          [env: ETH_GAS_PRICE=]

      --priority-gas-price <PRICE>
          Max priority fee per gas for EIP1559 transactions

          [env: ETH_PRIORITY_GAS_PRICE=]

      --value <VALUE>
          Ether to send in the transaction, either specified in wei, or as a
string with a unit
          type.

          Examples: 1ether, 10gwei, 0.01ether

      --nonce <NONCE>
          Nonce for the transaction

      --legacy
```

```
          Send a legacy transaction instead of an EIP1559 transaction.

          This is automatically enabled for common networks without EIP1559.

Ethereum options:
  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -e, --etherscan-api-key <KEY>
          The Etherscan (or equivalent) API key

          [env: ETHERSCAN_API_KEY=]

  -c, --chain <CHAIN>
          The chain name or EIP-155 chain ID

          [env: CHAIN=]

Wallet options - raw:
  -f, --from <ADDRESS>
          The sender account

          [env: ETH_FROM=]

  -i, --interactive
          Open an interactive prompt to enter your private key

      --private-key <RAW_PRIVATE_KEY>
          Use the provided private key

      --mnemonic <MNEMONIC>
          Use the mnemonic phrase of mnemonic file at the specified path
```

```
      --mnemonic-passphrase <PASSPHRASE>
          Use a BIP39 passphrase for the mnemonic

      --mnemonic-derivation-path <PATH>
          The wallet derivation path.

          Works with both --mnemonic-path and hardware wallets.

      --mnemonic-index <INDEX>
          Use the private key from the given mnemonic index.

          Used with --mnemonic-path.

          [default: 0]

Wallet options - keystore:
      --keystore <PATH>
          Use the keystore in the given folder or file

          [env: ETH_KEYSTORE=]

      --account <ACCOUNT_NAME>
          Use a keystore from the default keystores folder
(~/.foundry/keystores) by its filename

          [env: ETH_KEYSTORE_ACCOUNT=]

      --password <PASSWORD>
          The keystore password.

          Used with --keystore.

      --password-file <PASSWORD_FILE>
          The keystore password file path.

          Used with --keystore.

          [env: ETH_PASSWORD=]

Wallet options - hardware wallet:
  -l, --ledger
          Use a Ledger hardware wallet

  -t, --trezor
          Use a Trezor hardware wallet

Wallet options - AWS KMS:
      --aws
          Use AWS Key Management Service
```

cast address-zero

Prints the zero address

```
$ cast address-zero --help
Usage: cast address-zero

Options:
  -h, --help  Print help
```

cast admin

Fetch the EIP-1967 admin account

```
$ cast admin --help
Usage: cast admin [OPTIONS] <WHO>

Arguments:
  <WHO>
          The address to get the nonce for

Options:
  -B, --block <BLOCK>
          The block height to query at.

          Can also be the tags earliest, finalized, safe, latest, or pending.

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast age

Get the timestamp of a block

```
$ cast age --help
Usage: cast age [OPTIONS] [BLOCK]

Arguments:
  [BLOCK]
          The block height to query at.

          Can also be the tags earliest, finalized, safe, latest, or pending.

Options:
  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast balance

Get the balance of an account in wei

```
$ cast balance --help
Usage: cast balance [OPTIONS] <WHO>

Arguments:
  <WHO>
          The account to query

Options:
  -B, --block <BLOCK>
          The block height to query at.

          Can also be the tags earliest, finalized, safe, latest, or pending.

  -e, --ether
          Format the balance in ether

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

      --erc20 <ERC20>
          erc20 address to query, with the method `balanceOf(address) return
(uint256)`, alias with
          '--erc721'

  -h, --help
          Print help (see a summary with '-h')
```

cast base-fee

Get the basefee of a block

```
$ cast base-fee --help
Usage: cast base-fee [OPTIONS] [BLOCK]

Arguments:
  [BLOCK]
          The block height to query at.

          Can also be the tags earliest, finalized, safe, latest, or pending.

Options:
  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast bind

Generate a rust binding from a given ABI

```
$ cast bind --help
Usage: cast bind [OPTIONS] <PATH_OR_ADDRESS>

Arguments:
  <PATH_OR_ADDRESS>
          The contract address, or the path to an ABI Directory

          If an address is specified, then the ABI is fetched from Etherscan.

Options:
  -o, --output-dir <PATH>
          Path to where bindings will be stored

      --crate-name <NAME>
          The name of the Rust crate to generate.

          This should be a valid crates.io crate name. However, this is
currently not validated by
          this command.

          [default: foundry-contracts]

      --crate-version <VERSION>
          The version of the Rust crate to generate.

          This should be a standard semver version string. However, it is not
currently validated by
          this command.

          [default: 0.0.1]

      --separate-files
          Generate bindings as separate files

  -e, --etherscan-api-key <KEY>
          The Etherscan (or equivalent) API key

          [env: ETHERSCAN_API_KEY=]

  -c, --chain <CHAIN>
          The chain name or EIP-155 chain ID

          [env: CHAIN=]

  -h, --help
          Print help (see a summary with '-h')
```

cast block

Get information about a block

```
$ cast block --help
Usage: cast block [OPTIONS] [BLOCK]

Arguments:
  [BLOCK]
          The block height to query at.

          Can also be the tags earliest, finalized, safe, latest, or pending.

Options:
  -f, --field <FIELD>
          If specified, only get the given field of the block

      --full
          [env: CAST_FULL_BLOCK=]

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')

Display options:
  -j, --json
          Print the block as JSON
```

cast block-number

Get the latest block number

```
$ cast block-number --help
Usage: cast block-number [OPTIONS]

Options:
  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast call

Perform a call on an account without publishing a transaction

```
$ cast call --help
Usage: cast call [OPTIONS] [TO] [SIG] [ARGS]... [COMMAND]

Commands:
  --create  ignores the address field and simulates creating a contract
  help      Print this message or the help of the given subcommand(s)

Arguments:
  [TO]
          The destination of the transaction

  [SIG]
          The signature of the function to call

  [ARGS]...
          The arguments of the function to call

Options:
      --data <DATA>
          Data for the transaction

      --trace
          Forks the remote rpc, executes the transaction locally and prints a
trace

      --debug
          Opens an interactive debugger. Can only be used with `--trace`

      --labels <LABELS>
          Labels to apply to the traces; format: `address:label`. Can only be
used with `--trace`

      --evm-version <EVM_VERSION>
          The EVM Version to use. Can only be used with `--trace`

  -b, --block <BLOCK>
          The block height to query at.

          Can also be the tags earliest, finalized, safe, latest, or pending.

  -h, --help
          Print help (see a summary with '-h')

Transaction options:
      --gas-limit <GAS_LIMIT>
          Gas limit for the transaction

          [env: ETH_GAS_LIMIT=]

      --gas-price <PRICE>
          Gas price for legacy transactions, or max fee per gas for EIP1559
transactions

          [env: ETH_GAS_PRICE=]

      --priority-gas-price <PRICE>
          Max priority fee per gas for EIP1559 transactions
```

```
          [env: ETH_PRIORITY_GAS_PRICE=]

      --value <VALUE>
          Ether to send in the transaction, either specified in wei, or as a
string with a unit
          type.

          Examples: 1ether, 10gwei, 0.01ether

      --nonce <NONCE>
          Nonce for the transaction

      --legacy
          Send a legacy transaction instead of an EIP1559 transaction.

          This is automatically enabled for common networks without EIP1559.

Ethereum options:
  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -e, --etherscan-api-key <KEY>
          The Etherscan (or equivalent) API key

          [env: ETHERSCAN_API_KEY=]

  -c, --chain <CHAIN>
          The chain name or EIP-155 chain ID

          [env: CHAIN=]

Wallet options - raw:
```

```
  -f, --from <ADDRESS>
          The sender account

          [env: ETH_FROM=]

  -i, --interactive
          Open an interactive prompt to enter your private key

      --private-key <RAW_PRIVATE_KEY>
          Use the provided private key

      --mnemonic <MNEMONIC>
          Use the mnemonic phrase of mnemonic file at the specified path

      --mnemonic-passphrase <PASSPHRASE>
          Use a BIP39 passphrase for the mnemonic

      --mnemonic-derivation-path <PATH>
          The wallet derivation path.

          Works with both --mnemonic-path and hardware wallets.

      --mnemonic-index <INDEX>
          Use the private key from the given mnemonic index.

          Used with --mnemonic-path.

          [default: 0]

Wallet options - keystore:
      --keystore <PATH>
          Use the keystore in the given folder or file

          [env: ETH_KEYSTORE=]

      --account <ACCOUNT_NAME>
          Use a keystore from the default keystores folder
(~/.foundry/keystores) by its filename

          [env: ETH_KEYSTORE_ACCOUNT=]

      --password <PASSWORD>
          The keystore password.

          Used with --keystore.

      --password-file <PASSWORD_FILE>
          The keystore password file path.

          Used with --keystore.

          [env: ETH_PASSWORD=]

Wallet options - hardware wallet:
  -l, --ledger
          Use a Ledger hardware wallet

  -t, --trezor
```

```
            Use a Trezor hardware wallet

Wallet options - AWS KMS:
      --aws
            Use AWS Key Management Service
```

cast call –create

ignores the address field and simulates creating a contract

```
$ cast call --create --help
Usage: cast call --create [OPTIONS] <CODE> [SIG] [ARGS]...

Arguments:
  <CODE>
          Bytecode of contract

  [SIG]
          The signature of the constructor

  [ARGS]...
          The arguments of the constructor

Options:
      --value <VALUE>
          Ether to send in the transaction.

          Either specified in wei, or as a string with a unit type.

          Examples: 1ether, 10gwei, 0.01ether

  -h, --help
          Print help (see a summary with '-h')
```

cast calldata

ABI-encode a function with arguments

```
$ cast calldata --help
Usage: cast calldata <SIG> [ARGS]...

Arguments:
  <SIG>      The function signature in the format `<name>(<in-types>)(<out-
types>)`
  [ARGS]...  The arguments to encode

Options:
  -h, --help  Print help
```

cast calldata-decode

Decode ABI-encoded input data.

```
$ cast calldata-decode --help
Usage: cast calldata-decode <SIG> <CALLDATA>

Arguments:
  <SIG>
          The function signature in the format `<name>(<in-types>)(<out-
types>)`

  <CALLDATA>
          The ABI-encoded calldata

Options:
  -h, --help
          Print help (see a summary with '-h')
```

cast chain

Get the symbolic name of the current chain

```
$ cast chain --help
Usage: cast chain [OPTIONS]

Options:
  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast chain-id

Get the Ethereum chain ID

```
$ cast chain-id --help
Usage: cast chain-id [OPTIONS]

Options:
  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast client

Get the current client version

```
$ cast client --help
Usage: cast client [OPTIONS]

Options:
  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast code

Get the runtime bytecode of a contract

```
$ cast code --help
Usage: cast code [OPTIONS] <WHO>

Arguments:
  <WHO>
          The contract address

Options:
  -B, --block <BLOCK>
          The block height to query at.

          Can also be the tags earliest, finalized, safe, latest, or pending.

  -d, --disassemble
          Disassemble bytecodes into individual opcodes

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast codesize

Get the runtime bytecode size of a contract

```
$ cast codesize --help
Usage: cast codesize [OPTIONS] <WHO>

Arguments:
  <WHO>
          The contract address

Options:
  -B, --block <BLOCK>
          The block height to query at.

          Can also be the tags earliest, finalized, safe, latest, or pending.

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast completions

Generate shell completions script

```
$ cast completions --help
Usage: cast completions <SHELL>

Arguments:
  <SHELL>  [possible values: bash, elvish, fish, powershell, zsh]

Options:
  -h, --help  Print help
```

cast compute-address

Compute the contract address from a given nonce and deployer address

```
$ cast compute-address --help
Usage: cast compute-address [OPTIONS] [ADDRESS]

Arguments:
  [ADDRESS]
          The deployer address

Options:
      --nonce <NONCE>
          The nonce of the deployer address

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast concat-hex

Concatenate hex strings

```
$ cast concat-hex --help
Usage: cast concat-hex [DATA]...

Arguments:
  [DATA]...  The data to concatenate

Options:
  -h, --help  Print help
```

cast create2

Generate a deterministic contract address using CREATE2

```
$ cast create2 --help
Usage: cast create2 [OPTIONS]

Options:
  -s, --starts-with <HEX>      Prefix for the contract address
  -e, --ends-with <HEX>        Suffix for the contract address
  -m, --matching <HEX>         Sequence that the address has to match
  -c, --case-sensitive         Case sensitive matching
  -d, --deployer <ADDRESS>     Address of the contract deployer [default:
                               0x4e59b44847b379578588920ca78fbf26c0b4956c]
  -i, --init-code <HEX>        Init code of the contract to be deployed
      --init-code-hash <HASH>  Init code hash of the contract to be deployed
  -j, --jobs <JOBS>            Number of threads to use. Defaults to and caps
at the number of
                               logical cores
      --caller <ADDRESS>       Address of the caller. Used for the first 20
bytes of the salt
      --seed <HEX>             The random number generator's seed, used to
initialize the salt
      --no-random              Don't initialize the salt with a random value,
and instead use the
                               default value of 0
  -h, --help                   Print help
```

cast decode-transaction

Decodes a raw signed EIP 2718 typed transaction

```
$ cast decode-transaction --help
Usage: cast decode-transaction [TX]

Arguments:
  [TX]

Options:
  -h, --help  Print help
```

cast disassemble

Disassembles hex encoded bytecode into individual / human readable opcodes

```
$ cast disassemble --help
Usage: cast disassemble <BYTECODE>

Arguments:
  <BYTECODE>  The hex encoded bytecode

Options:
  -h, --help  Print help
```

cast estimate

Estimate the gas cost of a transaction

```
$ cast estimate --help
Usage: cast estimate [OPTIONS] [TO] [SIG] [ARGS]... [COMMAND]

Commands:
  --create  Estimate gas cost to deploy a smart contract
  help      Print this message or the help of the given subcommand(s)

Arguments:
  [TO]
          The destination of the transaction

  [SIG]
          The signature of the function to call

  [ARGS]...
          The arguments of the function to call

Options:
  -f, --from <FROM>
          The sender account

          [env: ETH_FROM=]
          [default: 0x0000000000000000000000000000000000000000]

      --value <VALUE>
          Ether to send in the transaction.

          Either specified in wei, or as a string with a unit type:

          Examples: 1ether, 10gwei, 0.01ether

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'
```

```
            [env: ETH_RPC_JWT_SECRET=]

      -e, --etherscan-api-key <KEY>
            The Etherscan (or equivalent) API key

            [env: ETHERSCAN_API_KEY=]

      -c, --chain <CHAIN>
            The chain name or EIP-155 chain ID

            [env: CHAIN=]

      -h, --help
            Print help (see a summary with '-h')
```

cast estimate –create

Estimate gas cost to deploy a smart contract

```
$ cast estimate --create --help
Usage: cast estimate --create [OPTIONS] <CODE> [SIG] [ARGS]...

Arguments:
  <CODE>
          The bytecode of contract

  [SIG]
          The signature of the constructor

  [ARGS]...
          Constructor arguments

Options:
      --value <VALUE>
          Ether to send in the transaction

          Either specified in wei, or as a string with a unit type:

          Examples: 1ether, 10gwei, 0.01ether

  -h, --help
          Print help (see a summary with '-h')
```

cast etherscan-source

Get the source code of a contract from Etherscan

```
$ cast etherscan-source --help
Usage: cast etherscan-source [OPTIONS] <ADDRESS>

Arguments:
  <ADDRESS>  The contract's address

Options:
  -d <DIRECTORY>                 The output directory to expand source tree
into
  -e, --etherscan-api-key <KEY>  The Etherscan (or equivalent) API key [env:
ETHERSCAN_API_KEY=]
  -c, --chain <CHAIN>            The chain name or EIP-155 chain ID [env:
CHAIN=]
  -h, --help                     Print help
```

cast find-block

Get the block number closest to the provided timestamp

```
$ cast find-block --help
Usage: cast find-block [OPTIONS] <TIMESTAMP>

Arguments:
  <TIMESTAMP>
          The UNIX timestamp to search for, in seconds

Options:
  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast format-bytes32-string

Formats a string into bytes32 encoding

```
$ cast format-bytes32-string --help
Usage: cast format-bytes32-string [STRING]

Arguments:
  [STRING]  The string to format

Options:
  -h, --help  Print help
```

cast from-bin

Convert binary data into hex data

```
$ cast from-bin --help
Usage: cast from-bin

Options:
  -h, --help  Print help
```

cast from-fixed-point

Convert a fixed point number into an integer

```
$ cast from-fixed-point --help
Usage: cast from-fixed-point [DECIMALS] [VALUE]

Arguments:
  [DECIMALS]  The number of decimals to use
  [VALUE]     The value to convert

Options:
  -h, --help  Print help
```

cast from-rlp

Decodes RLP encoded data.

```
$ cast from-rlp --help
Usage: cast from-rlp [VALUE]

Arguments:
  [VALUE]
          The value to convert

Options:
  -h, --help
          Print help (see a summary with '-h')
```

cast from-utf8

Convert UTF8 text to hex

```
$ cast from-utf8 --help
Usage: cast from-utf8 [TEXT]

Arguments:
  [TEXT]  The text to convert

Options:
  -h, --help  Print help
```

cast from-wei

Convert wei into an ETH amount.

```
$ cast from-wei --help
Usage: cast from-wei [VALUE] [UNIT]

Arguments:
  [VALUE]
          The value to convert

  [UNIT]
          The unit to convert from (ether, gwei, wei)

          [default: eth]

Options:
  -h, --help
          Print help (see a summary with '-h')
```

cast gas-price

Get the current gas price

```
$ cast gas-price --help
Usage: cast gas-price [OPTIONS]

Options:
  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast generate-fig-spec

Generate Fig autocompletion spec

```
$ cast generate-fig-spec --help
Usage: cast generate-fig-spec

Options:
  -h, --help  Print help
```

cast hash-zero

Prints the zero hash

```
$ cast hash-zero --help
Usage: cast hash-zero

Options:
  -h, --help  Print help
```

cast implementation

Fetch the EIP-1967 implementation account

```
$ cast implementation --help
Usage: cast implementation [OPTIONS] <WHO>

Arguments:
  <WHO>
          The address to get the nonce for

Options:
  -B, --block <BLOCK>
          The block height to query at.

          Can also be the tags earliest, finalized, safe, latest, or pending.

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast index

Compute the storage slot for an entry in a mapping

```
$ cast index --help
Usage: cast index <KEY_TYPE> <KEY> <SLOT_NUMBER>

Arguments:
  <KEY_TYPE>     The mapping key type
  <KEY>          The mapping key
  <SLOT_NUMBER>  The storage slot of the mapping

Options:
  -h, --help  Print help
```

cast interface

Generate a Solidity interface from a given ABI.

```
$ cast interface --help
Usage: cast interface [OPTIONS] <PATH_OR_ADDRESS>

Arguments:
  <PATH_OR_ADDRESS>
          The contract address, or the path to an ABI file.

          If an address is specified, then the ABI is fetched from Etherscan.

Options:
  -n, --name <NAME>
          The name to use for the generated interface

  -p, --pragma <VERSION>
          Solidity pragma version

          [default: ^0.8.4]

  -o, --output <PATH>
          The path to the output file.

          If not specified, the interface will be output to stdout.

  -j, --json
          If specified, the interface will be output as JSON rather than
Solidity

  -e, --etherscan-api-key <KEY>
          The Etherscan (or equivalent) API key

          [env: ETHERSCAN_API_KEY=]

  -c, --chain <CHAIN>
          The chain name or EIP-155 chain ID

          [env: CHAIN=]

  -h, --help
          Print help (see a summary with '-h')
```

cast keccak

Hash arbitrary data using Keccak-256

```
$ cast keccak --help
Usage: cast keccak [DATA]

Arguments:
  [DATA]  The data to hash

Options:
  -h, --help  Print help
```

cast logs

Get logs by signature or topic

```
$ cast logs --help
Usage: cast logs [OPTIONS] [SIG_OR_TOPIC] [TOPICS_OR_ARGS]...

Arguments:
  [SIG_OR_TOPIC]
          The signature of the event to filter logs by which will be converted
to the first topic or
          a topic to filter on

  [TOPICS_OR_ARGS]...
          If used with a signature, the indexed fields of the event to filter
by. Otherwise, the
          remaining topics of the filter

Options:
      --from-block <FROM_BLOCK>
          The block height to start query at.

          Can also be the tags earliest, finalized, safe, latest, or pending.

      --to-block <TO_BLOCK>
          The block height to stop query at.

          Can also be the tags earliest, finalized, safe, latest, or pending.

      --address <ADDRESS>
          The contract address to filter on

      --subscribe
          If the RPC type and endpoints supports `eth_subscribe` stream logs
instead of printing and
          exiting. Will continue until interrupted or TO_BLOCK is reached

  -h, --help
          Print help (see a summary with '-h')

Display options:
  -j, --json
          Print the logs as JSON.s

Ethereum options:
  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
```

```
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -e, --etherscan-api-key <KEY>
          The Etherscan (or equivalent) API key

          [env: ETHERSCAN_API_KEY=]

  -c, --chain <CHAIN>
          The chain name or EIP-155 chain ID

          [env: CHAIN=]

Wallet options - raw:
  -f, --from <ADDRESS>
          The sender account

          [env: ETH_FROM=]

  -i, --interactive
          Open an interactive prompt to enter your private key

      --private-key <RAW_PRIVATE_KEY>
          Use the provided private key

      --mnemonic <MNEMONIC>
          Use the mnemonic phrase of mnemonic file at the specified path

      --mnemonic-passphrase <PASSPHRASE>
          Use a BIP39 passphrase for the mnemonic

      --mnemonic-derivation-path <PATH>
          The wallet derivation path.

          Works with both --mnemonic-path and hardware wallets.

      --mnemonic-index <INDEX>
          Use the private key from the given mnemonic index.

          Used with --mnemonic-path.

          [default: 0]

Wallet options - keystore:
      --keystore <PATH>
          Use the keystore in the given folder or file

          [env: ETH_KEYSTORE=]
```

```
      --account <ACCOUNT_NAME>
          Use a keystore from the default keystores folder
(~/.foundry/keystores) by its filename

          [env: ETH_KEYSTORE_ACCOUNT=]

      --password <PASSWORD>
          The keystore password.

          Used with --keystore.

      --password-file <PASSWORD_FILE>
          The keystore password file path.

          Used with --keystore.

          [env: ETH_PASSWORD=]

Wallet options - hardware wallet:
  -l, --ledger
          Use a Ledger hardware wallet

  -t, --trezor
          Use a Trezor hardware wallet

Wallet options - AWS KMS:
      --aws
          Use AWS Key Management Service
```

cast lookup-address

Perform an ENS reverse lookup

```
$ cast lookup-address --help
Usage: cast lookup-address [OPTIONS] [WHO]

Arguments:
  [WHO]
          The account to perform the lookup for

Options:
  -v, --verify
          Perform a normal lookup to verify that the address is correct

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast max-int

Prints the maximum value of the given integer type

```
$ cast max-int --help
Usage: cast max-int [TYPE]

Arguments:
  [TYPE]  The integer type to get the maximum value of [default: int256]

Options:
  -h, --help  Print help
```

cast max-uint

Prints the maximum value of the given integer type

```
$ cast max-uint --help
Usage: cast max-uint [TYPE]

Arguments:
  [TYPE]  The unsigned integer type to get the maximum value of [default:
uint256]

Options:
  -h, --help  Print help
```

cast min-int

Prints the minimum value of the given integer type

```
$ cast min-int --help
Usage: cast min-int [TYPE]

Arguments:
  [TYPE]  The integer type to get the minimum value of [default: int256]

Options:
  -h, --help  Print help
```

cast mktx

Build and sign a transaction

```
$ cast mktx --help
Usage: cast mktx [OPTIONS] [TO] [SIG] [ARGS]... [COMMAND]

Commands:
  --create  Use to deploy raw contract bytecode
  help      Print this message or the help of the given subcommand(s)

Arguments:
  [TO]
          The destination of the transaction.

          If not provided, you must use `cast mktx --create`.

  [SIG]
          The signature of the function to call

  [ARGS]...
          The arguments of the function to call

Options:
      --resend
          Reuse the latest nonce for the sender account

  -h, --help
          Print help (see a summary with '-h')

Transaction options:
      --gas-limit <GAS_LIMIT>
          Gas limit for the transaction

          [env: ETH_GAS_LIMIT=]

      --gas-price <PRICE>
          Gas price for legacy transactions, or max fee per gas for EIP1559
transactions

          [env: ETH_GAS_PRICE=]

      --priority-gas-price <PRICE>
          Max priority fee per gas for EIP1559 transactions

          [env: ETH_PRIORITY_GAS_PRICE=]

      --value <VALUE>
          Ether to send in the transaction, either specified in wei, or as a
string with a unit
          type.

          Examples: 1ether, 10gwei, 0.01ether

      --nonce <NONCE>
          Nonce for the transaction

      --legacy
          Send a legacy transaction instead of an EIP1559 transaction.

          This is automatically enabled for common networks without EIP1559.
```

```
Ethereum options:
  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -e, --etherscan-api-key <KEY>
          The Etherscan (or equivalent) API key

          [env: ETHERSCAN_API_KEY=]

  -c, --chain <CHAIN>
          The chain name or EIP-155 chain ID

          [env: CHAIN=]

Wallet options - raw:
  -f, --from <ADDRESS>
          The sender account

          [env: ETH_FROM=]

  -i, --interactive
          Open an interactive prompt to enter your private key

      --private-key <RAW_PRIVATE_KEY>
          Use the provided private key

      --mnemonic <MNEMONIC>
          Use the mnemonic phrase of mnemonic file at the specified path

      --mnemonic-passphrase <PASSPHRASE>
          Use a BIP39 passphrase for the mnemonic
```

```
        --mnemonic-derivation-path <PATH>
            The wallet derivation path.

            Works with both --mnemonic-path and hardware wallets.

        --mnemonic-index <INDEX>
            Use the private key from the given mnemonic index.

            Used with --mnemonic-path.

            [default: 0]

    Wallet options - keystore:
        --keystore <PATH>
            Use the keystore in the given folder or file

            [env: ETH_KEYSTORE=]

        --account <ACCOUNT_NAME>
            Use a keystore from the default keystores folder
    (~/.foundry/keystores) by its filename

            [env: ETH_KEYSTORE_ACCOUNT=]

        --password <PASSWORD>
            The keystore password.

            Used with --keystore.

        --password-file <PASSWORD_FILE>
            The keystore password file path.

            Used with --keystore.

            [env: ETH_PASSWORD=]

    Wallet options - hardware wallet:
      -l, --ledger
            Use a Ledger hardware wallet

      -t, --trezor
            Use a Trezor hardware wallet

    Wallet options - AWS KMS:
        --aws
            Use AWS Key Management Service
```

cast mktx –create

Use to deploy raw contract bytecode

```
$ cast mktx --create --help
Usage: cast mktx --create <CODE> [SIG] [ARGS]...

Arguments:
  <CODE>     The initialization bytecode of the contract to deploy
  [SIG]      The signature of the constructor
  [ARGS]...  The constructor arguments

Options:
  -h, --help  Print help
```

cast namehash

Calculate the ENS namehash of a name

```
$ cast namehash --help
Usage: cast namehash [NAME]

Arguments:
  [NAME]

Options:
  -h, --help  Print help
```

cast nonce

Get the nonce for an account

```
$ cast nonce --help
Usage: cast nonce [OPTIONS] <WHO>

Arguments:
  <WHO>
          The address to get the nonce for

Options:
  -B, --block <BLOCK>
          The block height to query at.

          Can also be the tags earliest, finalized, safe, latest, or pending.

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast parse-bytes32-address

Parses a checksummed address from bytes32 encoding.

```
$ cast parse-bytes32-address --help
Usage: cast parse-bytes32-address [BYTES]

Arguments:
  [BYTES]

Options:
  -h, --help  Print help
```

cast parse-bytes32-string

Parses a string from bytes32 encoding

```
$ cast parse-bytes32-string --help
Usage: cast parse-bytes32-string [BYTES]

Arguments:
  [BYTES]  The string to parse

Options:
  -h, --help  Print help
```

cast pretty-calldata

Pretty print calldata.

```
$ cast pretty-calldata --help
Usage: cast pretty-calldata [OPTIONS] [CALLDATA]

Arguments:
  [CALLDATA]
          The calldata

Options:
  -o, --offline
          Skip the https://openchain.xyz lookup

  -h, --help
          Print help (see a summary with '-h')
```

cast proof

Generate a storage proof for a given storage slot

```
$ cast proof --help
Usage: cast proof [OPTIONS] <ADDRESS> [SLOTS]...

Arguments:
  <ADDRESS>
          The contract address

  [SLOTS]...
          The storage slot numbers (hex or decimal)

Options:
  -B, --block <BLOCK>
          The block height to query at.

          Can also be the tags earliest, finalized, safe, latest, or pending.

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast publish

Publish a raw transaction to the network

```
$ cast publish --help
Usage: cast publish [OPTIONS] <RAW_TX>

Arguments:
  <RAW_TX>
          The raw transaction

Options:
      --async
          Only print the transaction hash and exit immediately

          [env: CAST_ASYNC=]

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast receipt

Get the transaction receipt for a transaction

```
$ cast receipt --help
Usage: cast receipt [OPTIONS] <TX_HASH> [FIELD]

Arguments:
  <TX_HASH>
          The transaction hash

  [FIELD]
          If specified, only get the given field of the transaction

Options:
      --confirmations <CONFIRMATIONS>
          The number of confirmations until the receipt is fetched

          [default: 1]

      --async
          Exit immediately if the transaction was not found

          [env: CAST_ASYNC=]

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')

Display options:
  -j, --json
          Print as JSON
```

cast resolve-name

Perform an ENS lookup

```
$ cast resolve-name --help
Usage: cast resolve-name [OPTIONS] [WHO]

Arguments:
  [WHO]
          The name to lookup

Options:
  -v, --verify
          Perform a reverse lookup to verify that the name is correct

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast rpc

Perform a raw JSON-RPC request

```
$ cast rpc --help
Usage: cast rpc [OPTIONS] <METHOD> [PARAMS]...

Arguments:
  <METHOD>
          RPC method name

  [PARAMS]...
          RPC parameters

          Interpreted as JSON:

          cast rpc eth_getBlockByNumber 0x123 false => {"method":
"eth_getBlockByNumber", "params":
          ["0x123", false] ... }

Options:
  -w, --raw
          Send raw JSON parameters

          The first param will be interpreted as a raw JSON array of params. If
no params are given,
          stdin will be used. For example:

          cast rpc eth_getBlockByNumber '["0x123", false]' --raw => {"method":
          "eth_getBlockByNumber", "params": ["0x123", false] ... }

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')
```

cast run

Runs a published transaction in a local environment and prints the trace

```
$ cast run --help
Usage: cast run [OPTIONS] <TX_HASH>

Arguments:
  <TX_HASH>
          The transaction hash

Options:
  -d, --debug
          Opens the transaction in the debugger

  -q, --quick
          Executes the transaction only with the state from the previous block.

          May result in different results than the live execution!

  -v, --verbose
          Prints the full address of the contract

  -l, --label <LABEL>
          Label addresses in the trace.

          Example: 0xd8dA6BF26964aF9D7eEd9e03E53415D37aA96045:vitalik.eth

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -e, --evm-version <EVM_VERSION>
          The EVM version to use.

          Overrides the version specified in the config.
```

```
      --compute-units-per-second <CUPS>
          Sets the number of assumed available compute units per second for
this provider

          default value: 330

          See also,
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

      --no-rate-limit
          Disables rate limiting for this node's provider.

          default value: false

          See also,
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

          [aliases: no-rpc-rate-limit]

  -h, --help
          Print help (see a summary with '-h')
```

cast selectors

Extracts function selectors and arguments from bytecode

```
$ cast selectors --help
Usage: cast selectors [OPTIONS] <BYTECODE>

Arguments:
  <BYTECODE>  The hex encoded bytecode

Options:
  -r, --resolve  Resolve the function signatures for the extracted selectors
using
                 https://openchain.xyz
  -h, --help     Print help
```

cast send

Sign and publish a transaction

```
$ cast send --help
Usage: cast send [OPTIONS] [TO] [SIG] [ARGS]... [COMMAND]

Commands:
  --create  Use to deploy raw contract bytecode
  help      Print this message or the help of the given subcommand(s)

Arguments:
  [TO]
          The destination of the transaction.

          If not provided, you must use cast send --create.

  [SIG]
          The signature of the function to call

  [ARGS]...
          The arguments of the function to call

Options:
      --async
          Only print the transaction hash and exit immediately

          [env: CAST_ASYNC=]

      --confirmations <CONFIRMATIONS>
          The number of confirmations until the receipt is fetched

          [default: 1]

      --resend
          Reuse the latest nonce for the sender account

      --unlocked
          Send via `eth_sendTransaction using the `--from` argument or
$ETH_FROM as sender

  -h, --help
          Print help (see a summary with '-h')

Display options:
  -j, --json
          Print the transaction receipt as JSON

Transaction options:
      --gas-limit <GAS_LIMIT>
          Gas limit for the transaction

          [env: ETH_GAS_LIMIT=]

      --gas-price <PRICE>
          Gas price for legacy transactions, or max fee per gas for EIP1559
transactions

          [env: ETH_GAS_PRICE=]

      --priority-gas-price <PRICE>
```

```
            Max priority fee per gas for EIP1559 transactions

            [env: ETH_PRIORITY_GAS_PRICE=]

        --value <VALUE>
            Ether to send in the transaction, either specified in wei, or as a
string with a unit
            type.

            Examples: 1ether, 10gwei, 0.01ether

        --nonce <NONCE>
            Nonce for the transaction

        --legacy
            Send a legacy transaction instead of an EIP1559 transaction.

            This is automatically enabled for common networks without EIP1559.

Ethereum options:
  -r, --rpc-url <URL>
            The RPC endpoint

            [env: ETH_RPC_URL=]

        --flashbots
            Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
            transaction privately with all registered builders.
            https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

        --jwt-secret <JWT_SECRET>
            JWT Secret for the RPC endpoint.

            The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
            used to simulate a CL `engine_forkchoiceUpdated` call:

            cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
            "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

            [env: ETH_RPC_JWT_SECRET=]

  -e, --etherscan-api-key <KEY>
            The Etherscan (or equivalent) API key

            [env: ETHERSCAN_API_KEY=]

  -c, --chain <CHAIN>
            The chain name or EIP-155 chain ID

            [env: CHAIN=]
```

```
Wallet options - raw:
  -f, --from <ADDRESS>
          The sender account

          [env: ETH_FROM=]

  -i, --interactive
          Open an interactive prompt to enter your private key

      --private-key <RAW_PRIVATE_KEY>
          Use the provided private key

      --mnemonic <MNEMONIC>
          Use the mnemonic phrase of mnemonic file at the specified path

      --mnemonic-passphrase <PASSPHRASE>
          Use a BIP39 passphrase for the mnemonic

      --mnemonic-derivation-path <PATH>
          The wallet derivation path.

          Works with both --mnemonic-path and hardware wallets.

      --mnemonic-index <INDEX>
          Use the private key from the given mnemonic index.

          Used with --mnemonic-path.

          [default: 0]

Wallet options - keystore:
      --keystore <PATH>
          Use the keystore in the given folder or file

          [env: ETH_KEYSTORE=]

      --account <ACCOUNT_NAME>
          Use a keystore from the default keystores folder
(~/.foundry/keystores) by its filename

          [env: ETH_KEYSTORE_ACCOUNT=]

      --password <PASSWORD>
          The keystore password.

          Used with --keystore.

      --password-file <PASSWORD_FILE>
          The keystore password file path.

          Used with --keystore.

          [env: ETH_PASSWORD=]

Wallet options - hardware wallet:
  -l, --ledger
          Use a Ledger hardware wallet
```

```
      -t, --trezor
              Use a Trezor hardware wallet

    Wallet options - AWS KMS:
          --aws
              Use AWS Key Management Service
```

cast send –create

Use to deploy raw contract bytecode

```
$ cast send --create --help
Usage: cast send --create <CODE> [SIG] [ARGS]...

Arguments:
  <CODE>     The bytecode of the contract to deploy
  [SIG]      The signature of the function to call
  [ARGS]...  The arguments of the function to call

Options:
  -h, --help  Print help
```

cast shl

Perform a left shifting operation

```
$ cast shl --help
Usage: cast shl [OPTIONS] <VALUE> <BITS>

Arguments:
  <VALUE>  The value to shift
  <BITS>   The number of bits to shift

Options:
      --base-in <BASE_IN>    The input base
      --base-out <BASE_OUT>  The output base [default: 16]
  -h, --help                 Print help
```

cast shr

Perform a right shifting operation

```
$ cast shr --help
Usage: cast shr [OPTIONS] <VALUE> <BITS>

Arguments:
  <VALUE>  The value to shift
  <BITS>   The number of bits to shift

Options:
      --base-in <BASE_IN>    The input base,
      --base-out <BASE_OUT>  The output base, [default: 16]
  -h, --help                 Print help
```

cast sig

Get the selector for a function

```
$ cast sig --help
Usage: cast sig [SIG] [OPTIMIZE]

Arguments:
  [SIG]       The function signature, e.g. transfer(address,uint256)
  [OPTIMIZE]  Optimize signature to contain provided amount of leading zeroes
in selector

Options:
  -h, --help  Print help
```

cast sig-event

Generate event signatures from event string

```
$ cast sig-event --help
Usage: cast sig-event [EVENT_STRING]

Arguments:
  [EVENT_STRING]  The event string

Options:
  -h, --help  Print help
```

cast storage

Get the raw value of a contract's storage slot

```
$ cast storage --help
Usage: cast storage [OPTIONS] <ADDRESS> [SLOT]

Arguments:
  <ADDRESS>
          The contract address

  [SLOT]
          The storage slot number

Options:
  -b, --block <BLOCK>
          The block height to query at.

          Can also be the tags earliest, finalized, safe, latest, or pending.

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -e, --etherscan-api-key <KEY>
          The Etherscan (or equivalent) API key

          [env: ETHERSCAN_API_KEY=]

  -c, --chain <CHAIN>
          The chain name or EIP-155 chain ID

          [env: CHAIN=]

  -h, --help
          Print help (see a summary with '-h')
```

```
Cache options:
      --force
          Clear the cache and artifacts folder and recompile

Build options:
      --no-cache
          Disable the cache

Linker options:
      --libraries <LIBRARIES>
          Set pre-linked libraries

          [env: DAPP_LIBRARIES=]

Compiler options:
      --ignored-error-codes <ERROR_CODES>
          Ignore solc warnings by error code

      --deny-warnings
          Warnings will trigger a compiler error

      --no-auto-detect
          Do not auto-detect the `solc` version

      --use <SOLC_VERSION>
          Specify the solc version, or a path to a local solc, to build with.

          Valid values are in the format `x.y.z`, `solc:x.y.z` or
`path/to/solc`.

      --offline
          Do not access the network.

          Missing solc versions will not be installed.

      --via-ir
          Use the Yul intermediate representation compilation pipeline

      --silent
          Don't print anything on startup

      --ast
          Includes the AST as JSON in the compiler output

      --evm-version <VERSION>
          The target EVM version

      --optimize
          Activate the Solidity optimizer

      --optimizer-runs <RUNS>
          The number of optimizer runs

      --extra-output <SELECTOR>...
          Extra output to include in the contract's artifact.

          Example keys: evm.assembly, ewasm, ir, irOptimized, metadata
```

```
          For a full description, see
          https://docs.soliditylang.org/en/v0.8.13/using-the-
compiler.html#input-description

      --extra-output-files <SELECTOR>...
          Extra output to write to separate files.

          Valid values: metadata, ir, irOptimized, ewasm, evm.assembly

Project options:
  -o, --out <PATH>
          The path to the contract artifacts folder

      --revert-strings <REVERT>
          Revert string configuration.

          Possible values are "default", "strip" (remove), "debug" (Solidity-
generated revert
          strings) and "verboseDebug"

      --build-info
          Generate build info files

      --build-info-path <PATH>
          Output path to directory that build info files will be written to

      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
          The contracts source directory

  -R, --remappings <REMAPPINGS>
          The project's remappings

      --remappings-env <ENV>
          The project's remappings from the environment

      --cache-path <PATH>
          The path to the compiler cache

      --lib-paths <PATH>
          The path to the library folder

      --hardhat
          Use the Hardhat-style project layout.

          This is the same as using: `--contracts contracts --lib-paths
node_modules`.

          [aliases: hh]

      --config-path <FILE>
          Path to the config file
```

cast to-ascii

Convert hex data to an ASCII string

```
$ cast to-ascii --help
Usage: cast to-ascii [HEXDATA]

Arguments:
  [HEXDATA]  The hex data to convert

Options:
  -h, --help  Print help
```

cast to-base

Converts a number of one base to another

```
$ cast to-base --help
Usage: cast to-base [OPTIONS] [VALUE] [BASE]

Arguments:
  [VALUE]  The value to convert
  [BASE]   The output base

Options:
  -i, --base-in <BASE_IN>  The input base
  -h, --help               Print help
```

cast to-bytes32

Right-pads hex data to 32 bytes

```
$ cast to-bytes32 --help
Usage: cast to-bytes32 [BYTES]

Arguments:
  [BYTES]  The hex data to convert

Options:
  -h, --help  Print help
```

cast to-check-sum-address

Convert an address to a checksummed format (EIP-55)

```
$ cast to-check-sum-address --help
Usage: cast to-check-sum-address [ADDRESS]

Arguments:
  [ADDRESS]  The address to convert

Options:
  -h, --help  Print help
```

cast to-dec

Converts a number of one base to decimal

```
$ cast to-dec --help
Usage: cast to-dec [OPTIONS] [VALUE]

Arguments:
  [VALUE]  The value to convert

Options:
  -i, --base-in <BASE_IN>  The input base
  -h, --help               Print help
```

cast to-fixed-point

Convert an integer into a fixed point number

```
$ cast to-fixed-point --help
Usage: cast to-fixed-point [DECIMALS] [VALUE]

Arguments:
  [DECIMALS]  The number of decimals to use
  [VALUE]     The value to convert

Options:
  -h, --help  Print help
```

cast to-hex

Converts a number of one base to another

```
$ cast to-hex --help
Usage: cast to-hex [OPTIONS] [VALUE]

Arguments:
  [VALUE]  The value to convert

Options:
  -i, --base-in <BASE_IN>  The input base
  -h, --help               Print help
```

cast to-hexdata

Normalize the input to lowercase, 0x-prefixed hex.

```
$ cast to-hexdata --help
Usage: cast to-hexdata [INPUT]

Arguments:
  [INPUT]
          The input to normalize

Options:
  -h, --help
          Print help (see a summary with '-h')
```

cast to-int256

Convert a number to a hex-encoded int256

```
$ cast to-int256 --help
Usage: cast to-int256 [VALUE]

Arguments:
  [VALUE]  The value to convert

Options:
  -h, --help  Print help
```

cast to-rlp

RLP encodes hex data, or an array of hex data

```
$ cast to-rlp --help
Usage: cast to-rlp [VALUE]

Arguments:
  [VALUE]  The value to convert

Options:
  -h, --help  Print help
```

cast to-uint256

Convert a number to a hex-encoded uint256

```
$ cast to-uint256 --help
Usage: cast to-uint256 [VALUE]

Arguments:
  [VALUE]  The value to convert

Options:
  -h, --help  Print help
```

cast to-unit

Convert an ETH amount into another unit (ether, gwei or wei).

```
$ cast to-unit --help
Usage: cast to-unit [VALUE] [UNIT]

Arguments:
  [VALUE]
          The value to convert

  [UNIT]
          The unit to convert to (ether, gwei, wei)

          [default: wei]

Options:
  -h, --help
          Print help (see a summary with '-h')
```

cast to-wei

Convert an ETH amount to wei.

```
$ cast to-wei --help
Usage: cast to-wei [VALUE] [UNIT]

Arguments:
  [VALUE]
          The value to convert

  [UNIT]
          The unit to convert from (ether, gwei, wei)

          [default: eth]

Options:
  -h, --help
          Print help (see a summary with '-h')
```

cast tx

Get information about a transaction

```
$ cast tx --help
Usage: cast tx [OPTIONS] <TX_HASH> [FIELD]

Arguments:
  <TX_HASH>
          The transaction hash

  [FIELD]
          If specified, only get the given field of the transaction. If "raw",
the RLP encoded
          transaction will be printed

Options:
      --raw
          Print the raw RLP encoded transaction

  -r, --rpc-url <URL>
          The RPC endpoint

          [env: ETH_RPC_URL=]

      --flashbots
          Use the Flashbots RPC URL with fast mode
(https://rpc.flashbots.net/fast). This shares the
          transaction privately with all registered builders.
          https://docs.flashbots.net/flashbots-protect/quick-start#faster-
transactions

      --jwt-secret <JWT_SECRET>
          JWT Secret for the RPC endpoint.

          The JWT secret will be used to create a JWT for a RPC. For example,
the following can be
          used to simulate a CL `engine_forkchoiceUpdated` call:

          cast rpc --jwt-secret <JWT_SECRET> engine_forkchoiceUpdatedV2

'["0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",
          "0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc",

"0x6bb38c26db65749ab6e472080a3d20a2f35776494e72016d1e339593f21c59bc"]'

          [env: ETH_RPC_JWT_SECRET=]

  -h, --help
          Print help (see a summary with '-h')

Display options:
  -j, --json
          Print as JSON
```

cast upload-signature

Upload the given signatures to https://openchain.xyz.

```
$ cast upload-signature --help
Usage: cast upload-signature [SIGNATURES]...

Arguments:
  [SIGNATURES]...
          The signatures to upload.

          Prefix with 'function', 'event', or 'error'. Defaults to function if
no prefix given. Can
          also take paths to contract artifact JSON.

Options:
  -h, --help
          Print help (see a summary with '-h')
```

cast wallet

Wallet management utilities

```
$ cast wallet --help
Usage: cast wallet <COMMAND>

Commands:
  new                 Create a new random keypair [aliases: n]
  new-mnemonic        Generates a random BIP39 mnemonic phrase [aliases: nm]
  vanity              Generate a vanity address [aliases: va]
  address             Convert a private key to an address [aliases: a, addr]
  sign                Sign a message or typed data [aliases: s]
  verify              Verify the signature of a message [aliases: v]
  import              Import a private key into an encrypted keystore [aliases:
i]
  list                List all the accounts in the keystore default directory
[aliases: ls]
  derive-private-key  Derives private key from mnemonic [aliases: --derive-
private-key]
  help                Print this message or the help of the given subcommand(s)

Options:
  -h, --help  Print help
```

cast wallet new

Create a new random keypair

```
$ cast wallet new --help
Usage: cast wallet new [OPTIONS] [PATH]

Arguments:
  [PATH]
          If provided, then keypair will be written to an encrypted JSON
keystore

Options:
  -p, --password
          Triggers a hidden password prompt for the JSON keystore.

          Deprecated: prompting for a hidden password is now the default.

      --unsafe-password <PASSWORD>
          Password for the JSON keystore in cleartext.

          This is UNSAFE to use and we recommend using the --password.

          [env: CAST_PASSWORD=]

  -n, --number <NUMBER>
          Number of wallets to generate

          [default: 1]

  -j, --json
          Output generated wallets as JSON

  -h, --help
          Print help (see a summary with '-h')
```

cast wallet new-mnemonic

Generates a random BIP39 mnemonic phrase

```
$ cast wallet new-mnemonic --help
Usage: cast wallet new-mnemonic [OPTIONS]

Options:
  -w, --words <WORDS>        Number of words for the mnemonic [default: 12]
  -a, --accounts <ACCOUNTS>  Number of accounts to display [default: 1]
  -h, --help                 Print help
```

cast wallet vanity

Generate a vanity address

```
$ cast wallet vanity --help
Usage: cast wallet vanity [OPTIONS]

Options:
      --starts-with <HEX>
          Prefix for the vanity address

      --ends-with <HEX>
          Suffix for the vanity address

      --nonce <NONCE>
          Generate a vanity contract address created by the generated keypair
with the specified
          nonce

      --save-path <PATH>
          Path to save the generated vanity contract address to.

          If provided, the generated vanity addresses will appended to a JSON
array in the specified
          file.

  -h, --help
          Print help (see a summary with '-h')
```

cast wallet address

Convert a private key to an address

```
$ cast wallet address --help
Usage: cast wallet address [OPTIONS] [PRIVATE_KEY]

Arguments:
  [PRIVATE_KEY]
          If provided, the address will be derived from the specified private
key

Options:
  -h, --help
          Print help (see a summary with '-h')

Wallet options - raw:
  -f, --from <ADDRESS>
          The sender account

          [env: ETH_FROM=]

  -i, --interactive
          Open an interactive prompt to enter your private key

      --private-key <RAW_PRIVATE_KEY>
          Use the provided private key

      --mnemonic <MNEMONIC>
          Use the mnemonic phrase of mnemonic file at the specified path

      --mnemonic-passphrase <PASSPHRASE>
          Use a BIP39 passphrase for the mnemonic

      --mnemonic-derivation-path <PATH>
          The wallet derivation path.

          Works with both --mnemonic-path and hardware wallets.

      --mnemonic-index <INDEX>
          Use the private key from the given mnemonic index.

          Used with --mnemonic-path.

          [default: 0]

Wallet options - keystore:
      --keystore <PATH>
          Use the keystore in the given folder or file

          [env: ETH_KEYSTORE=]

      --account <ACCOUNT_NAME>
          Use a keystore from the default keystores folder
(~/.foundry/keystores) by its filename

          [env: ETH_KEYSTORE_ACCOUNT=]

      --password <PASSWORD>
          The keystore password.
```

```
            Used with --keystore.

        --password-file <PASSWORD_FILE>
            The keystore password file path.

            Used with --keystore.

            [env: ETH_PASSWORD=]

    Wallet options - hardware wallet:
      -l, --ledger
            Use a Ledger hardware wallet

      -t, --trezor
            Use a Trezor hardware wallet

    Wallet options - AWS KMS:
        --aws
            Use AWS Key Management Service
```

cast wallet sign

Sign a message or typed data

```
$ cast wallet sign --help
Usage: cast wallet sign [OPTIONS] <MESSAGE>

Arguments:
  <MESSAGE>
          The message, typed data, or hash to sign.

          Messages starting with 0x are expected to be hex encoded, which get
decoded before being
          signed.

          The message will be prefixed with the Ethereum Signed Message header
and hashed before
          signing, unless `--no-hash` is provided.

          Typed data can be provided as a json string or a file name. Use --
data flag to denote the
          message is a string of typed data. Use --data --from-file to denote
the message is a file
          name containing typed data. The data will be combined and hashed
using the EIP712
          specification before signing. The data should be formatted as JSON.

Options:
      --data
          Treat the message as JSON typed data

      --from-file
          Treat the message as a file containing JSON typed data. Requires `--
data`

      --no-hash
          Treat the message as a raw 32-byte hash and sign it directly without
hashing it again

  -h, --help
          Print help (see a summary with '-h')

Wallet options - raw:
  -f, --from <ADDRESS>
          The sender account

          [env: ETH_FROM=]

  -i, --interactive
          Open an interactive prompt to enter your private key

      --private-key <RAW_PRIVATE_KEY>
          Use the provided private key

      --mnemonic <MNEMONIC>
          Use the mnemonic phrase of mnemonic file at the specified path

      --mnemonic-passphrase <PASSPHRASE>
          Use a BIP39 passphrase for the mnemonic

      --mnemonic-derivation-path <PATH>
```

```
            The wallet derivation path.

            Works with both --mnemonic-path and hardware wallets.

        --mnemonic-index <INDEX>
            Use the private key from the given mnemonic index.

            Used with --mnemonic-path.

            [default: 0]

    Wallet options - keystore:
        --keystore <PATH>
            Use the keystore in the given folder or file

            [env: ETH_KEYSTORE=]

        --account <ACCOUNT_NAME>
            Use a keystore from the default keystores folder
    (~/.foundry/keystores) by its filename

            [env: ETH_KEYSTORE_ACCOUNT=]

        --password <PASSWORD>
            The keystore password.

            Used with --keystore.

        --password-file <PASSWORD_FILE>
            The keystore password file path.

            Used with --keystore.

            [env: ETH_PASSWORD=]

    Wallet options - hardware wallet:
      -l, --ledger
            Use a Ledger hardware wallet

      -t, --trezor
            Use a Trezor hardware wallet

    Wallet options - AWS KMS:
        --aws
            Use AWS Key Management Service
```

cast wallet verify

Verify the signature of a message

```
$ cast wallet verify --help
Usage: cast wallet verify --address <ADDRESS> <MESSAGE> <SIGNATURE>

Arguments:
  <MESSAGE>    The original message
  <SIGNATURE>  The signature to verify

Options:
  -a, --address <ADDRESS>  The address of the message signer
  -h, --help               Print help
```

cast wallet import

Import a private key into an encrypted keystore

```
$ cast wallet import --help
Usage: cast wallet import [OPTIONS] <ACCOUNT_NAME>

Arguments:
  <ACCOUNT_NAME>
          The name for the account in the keystore

Options:
  -k, --keystore-dir <KEYSTORE_DIR>
          If provided, keystore will be saved here instead of the default
keystores directory
          (~/.foundry/keystores)

      --unsafe-password <PASSWORD>
          Password for the JSON keystore in cleartext This is unsafe, we
recommend using the default
          hidden password prompt

          [env: CAST_UNSAFE_PASSWORD=]

  -h, --help
          Print help (see a summary with '-h')

Wallet options - raw:
  -i, --interactive
          Open an interactive prompt to enter your private key

      --private-key <RAW_PRIVATE_KEY>
          Use the provided private key

      --mnemonic <MNEMONIC>
          Use the mnemonic phrase of mnemonic file at the specified path

      --mnemonic-passphrase <PASSPHRASE>
          Use a BIP39 passphrase for the mnemonic

      --mnemonic-derivation-path <PATH>
          The wallet derivation path.

          Works with both --mnemonic-path and hardware wallets.

      --mnemonic-index <INDEX>
          Use the private key from the given mnemonic index.

          Used with --mnemonic-path.

          [default: 0]
```

cast wallet list

List all the accounts in the keystore default directory

```
$ cast wallet list --help
Usage: cast wallet list [OPTIONS]

Options:
      --dir [<DIR>]                List all the accounts in the keystore
directory. Default keystore
                                   directory is used if no path provided
  -l, --ledger                     List accounts from a Ledger hardware wallet
  -t, --trezor                     List accounts from a Trezor hardware wallet
      --aws                        List accounts from AWS KMS
      --all                        List all configured accounts
  -m, --max-senders <MAX_SENDERS>  Max number of addresses to display from
hardware wallets
                                   [default: 3]
  -h, --help                       Print help
```

cast wallet derive-private-key

Derives private key from mnemonic

```
$ cast wallet derive-private-key --help
Usage: cast wallet derive-private-key <MNEMONIC> [MNEMONIC_INDEX]

Arguments:
  <MNEMONIC>
  [MNEMONIC_INDEX]

Options:
  -h, --help  Print help
```

anvil

A fast local Ethereum development node

```
$ anvil --help
Usage: anvil [OPTIONS] [COMMAND]

Commands:
  completions        Generate shell completions script [aliases: com]
  generate-fig-spec  Generate Fig autocompletion spec [aliases: fig]
  help               Print this message or the help of the given subcommand(s)

Options:
  -a, --accounts <NUM>
          Number of dev accounts to generate and configure

          [default: 10]

  -b, --block-time <SECONDS>
          Block time in seconds for interval mining

          [aliases: blockTime]

      --balance <NUM>
          The balance of every dev account in Ether

          [default: 10000]

      --config-out <OUT_FILE>
          Writes output of `anvil` as json to user-specified file

      --derivation-path <DERIVATION_PATH>
          Sets the derivation path of the child key to be derived.

          [default: m/44'/60'/0'/0/]

      --dump-state <PATH>
          Dump the state and block environment of chain on exit to the given
file.

          If the value is a directory, the state will be written to
`<VALUE>/state.json`.

  -h, --help
          Print help (see a summary with '-h')

      --hardfork <HARDFORK>
          The EVM hardfork to use.

          Choose the hardfork by name, e.g. `shanghai`, `paris`, `london`,
etc... [default: latest]

      --init <PATH>
          Initialize the genesis block with the given `genesis.json` file

      --ipc [<PATH>]
          Launch an ipc server at the given path or default path =
`/tmp/anvil.ipc`

          [aliases: ipcpath]
```

```
      --load-state <PATH>
          Initialize the chain from a previously saved state snapshot

  -m, --mnemonic <MNEMONIC>
          BIP39 mnemonic phrase used for generating accounts. Cannot be used if
`mnemonic_random` or
          `mnemonic_seed` are used

      --mnemonic-random [<MNEMONIC_RANDOM>]
          Automatically generates a BIP39 mnemonic phrase, and derives accounts
from it. Cannot be
          used with other `mnemonic` options You can specify the number of
words you want in the
          mnemonic. [default: 12]

      --mnemonic-seed-unsafe <MNEMONIC_SEED>
          Generates a BIP39 mnemonic phrase from a given seed Cannot be used
with other `mnemonic`
          options

          CAREFUL: this is NOT SAFE and should only be used for testing. Never
use the private keys
          generated in production.

      --no-mining
          Disable auto and interval mining, and mine on demand instead

          [aliases: no-mine]

      --order <ORDER>
          How transactions are sorted in the mempool

          [default: fees]

  -p, --port <NUM>
          Port number to listen on

          [default: 8545]

      --prune-history [<PRUNE_HISTORY>]
          Don't keep full chain history. If a number argument is specified, at
most this number of
          states is kept in memory

  -s, --state-interval <SECONDS>
          Interval in seconds at which the state and block environment is to be
dumped to disk.

          See --state and --dump-state

      --silent
          Don't print anything on startup and don't print logs

      --slots-in-an-epoch <SLOTS_IN_AN_EPOCH>
          Slots in an epoch

          [default: 32]
```

```
      --state <PATH>
          This is an alias for both --load-state and --dump-state.

          It initializes the chain with the state and block environment stored
at the file, if it
          exists, and dumps the chain's state on exit.

      --timestamp <NUM>
          The timestamp of the genesis block

      --transaction-block-keeper <TRANSACTION_BLOCK_KEEPER>
          Number of blocks with transactions to keep in memory

  -V, --version
          Print version

Server options:
      --allow-origin <ALLOW_ORIGIN>
          Set the CORS allow_origin

          [default: *]

      --host <IP_ADDR>
          The hosts the server will listen on

          [env: ANVIL_IP_ADDR=]
          [default: 127.0.0.1]

      --no-cors
          Disable CORS

Fork config:
      --compute-units-per-second <CUPS>
          Sets the number of assumed available compute units per second for
this provider

          default value: 330

          See --fork-url. See also,
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

  -f, --fork-url <URL>
          Fetch state over a remote endpoint instead of starting from an empty
state.

          If you want to fetch state from a specific block number, add a block
number like
          `http://localhost:8545@1400000` or use the `--fork-block-number`
argument.

          [aliases: rpc-url]

      --fork-block-number <BLOCK>
          Fetch state from a specific block number over a remote endpoint.

          See --fork-url.
```

```
      --fork-chain-id <CHAIN>
          Specify chain id to skip fetching it from remote endpoint. This
enables offline-start
          mode.

          You still must pass both `--fork-url` and `--fork-block-number`, and
already have your
          required state cached on disk, anything missing locally would be
fetched from the remote.

      --fork-header <HEADERS>
          Headers to use for the rpc client, e.g. "User-Agent: test-agent"

          See --fork-url.

      --fork-retry-backoff <BACKOFF>
          Initial retry backoff on encountering errors.

          See --fork-url.

      --no-rate-limit
          Disables rate limiting for this node's provider.

          default value: false

          See --fork-url. See also,
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

          [aliases: no-rpc-rate-limit]

      --no-storage-caching
          Explicitly disables the use of RPC caching.

          All storage slots are read entirely from the endpoint.

          This flag overrides the project's configuration file.

          See --fork-url.

      --retries <retries>
          Number of retry requests for spurious networks (timed out requests)

          Default value 5

      --timeout <timeout>
          Timeout in ms for requests sent to remote JSON-RPC server in forking
mode.

          Default value 45000

Environment config:
      --block-base-fee-per-gas <FEE>
          The base fee in a block

          [aliases: base-fee]

      --chain-id <CHAIN_ID>
```

```
            The chain ID

      --code-size-limit <CODE_SIZE>
          EIP-170: Contract code size limit in bytes. Useful to increase this
because of tests. By
          default, it is 0x6000 (~25kb)

      --disable-block-gas-limit
          Disable the `call.gas_limit <= block.gas_limit` constraint

      --gas-limit <GAS_LIMIT>
          The block gas limit

      --gas-price <GAS_PRICE>
          The gas price

EVM options:
      --auto-impersonate
          Enable autoImpersonate on startup

          [aliases: auto-impersonate]

      --disable-default-create2-deployer
          Disable the default create2 deployer

          [aliases: no-create2]

      --optimism
          Run an Optimism chain

          [aliases: optimism]

      --steps-tracing
          Enable steps tracing used for debug calls returning geth-style traces

          [aliases: tracing]
```

anvil completions

Generate shell completions script

```
$ anvil completions --help
Usage: anvil completions <SHELL>

Arguments:
  <SHELL>  [possible values: bash, elvish, fish, powershell, zsh]

Options:
  -h, --help  Print help
```

anvil generate-fig-spec

Generate Fig autocompletion spec

```
$ anvil generate-fig-spec --help
Usage: anvil generate-fig-spec

Options:
  -h, --help  Print help
```

chisel

Fast, utilitarian, and verbose Solidity REPL

```
$ chisel --help
Usage: chisel [OPTIONS] [COMMAND]

Commands:
  list         List all cached sessions
  load         Load a cached session
  view         View the source of a cached session
  clear-cache  Clear all cached chisel sessions from the cache directory
  help         Print this message or the help of the given subcommand(s)

Options:
  -h, --help
          Print help (see a summary with '-h')

  -V, --version
          Print version

REPL options:
      --prelude <PRELUDE>
          Path to a directory containing Solidity files to import, or path to a
single Solidity
          file.

          These files will be evaluated before the top-level of the REPL,
therefore functioning as a
          prelude

      --no-vm
          Disable the default `Vm` import.

          The import is disabled by default if the Solc version is less than
0.6.2.

Cache options:
      --force
          Clear the cache and artifacts folder and recompile

Build options:
      --no-cache
          Disable the cache

Linker options:
      --libraries <LIBRARIES>
          Set pre-linked libraries

          [env: DAPP_LIBRARIES=]

Compiler options:
      --ignored-error-codes <ERROR_CODES>
          Ignore solc warnings by error code

      --deny-warnings
          Warnings will trigger a compiler error

      --no-auto-detect
          Do not auto-detect the `solc` version
```

```
      --use <SOLC_VERSION>
          Specify the solc version, or a path to a local solc, to build with.

          Valid values are in the format `x.y.z`, `solc:x.y.z` or
`path/to/solc`.

      --offline
          Do not access the network.

          Missing solc versions will not be installed.

      --via-ir
          Use the Yul intermediate representation compilation pipeline

      --silent
          Don't print anything on startup

      --ast
          Includes the AST as JSON in the compiler output

      --evm-version <VERSION>
          The target EVM version

      --optimize
          Activate the Solidity optimizer

      --optimizer-runs <RUNS>
          The number of optimizer runs

      --extra-output <SELECTOR>...
          Extra output to include in the contract's artifact.

          Example keys: evm.assembly, ewasm, ir, irOptimized, metadata

          For a full description, see
          https://docs.soliditylang.org/en/v0.8.13/using-the-
compiler.html#input-description

      --extra-output-files <SELECTOR>...
          Extra output to write to separate files.

          Valid values: metadata, ir, irOptimized, ewasm, evm.assembly

Project options:
  -o, --out <PATH>
          The path to the contract artifacts folder

      --revert-strings <REVERT>
          Revert string configuration.

          Possible values are "default", "strip" (remove), "debug" (Solidity-
generated revert
          strings) and "verboseDebug"

      --build-info
          Generate build info files

      --build-info-path <PATH>
```

```
            Output path to directory that build info files will be written to

      --root <PATH>
          The project's root path.

          By default root of the Git repository, if in one, or the current
working directory.

  -C, --contracts <PATH>
          The contracts source directory

  -R, --remappings <REMAPPINGS>
          The project's remappings

      --remappings-env <ENV>
          The project's remappings from the environment

      --cache-path <PATH>
          The path to the compiler cache

      --lib-paths <PATH>
          The path to the library folder

      --hardhat
          Use the Hardhat-style project layout.

          This is the same as using: `--contracts contracts --lib-paths
node_modules`.

          [aliases: hh]

      --config-path <FILE>
          Path to the config file

EVM options:
  -f, --fork-url <URL>
          Fetch state over a remote endpoint instead of starting from an empty
state.

          If you want to fetch state from a specific block number, see --fork-
block-number.

          [aliases: rpc-url]

      --fork-block-number <BLOCK>
          Fetch state from a specific block number over a remote endpoint.

          See --fork-url.

      --fork-retries <RETRIES>
          Number of retries.

          See --fork-url.

      --fork-retry-backoff <BACKOFF>
          Initial retry backoff on encountering errors.

          See --fork-url.
```

```
      --no-storage-caching
          Explicitly disables the use of RPC caching.

          All storage slots are read entirely from the endpoint.

          This flag overrides the project's configuration file.

          See --fork-url.

      --initial-balance <BALANCE>
          The initial balance of deployed test contracts

      --sender <ADDRESS>
          The address which will be executing tests

      --ffi
          Enable the FFI cheatcode

      --always-use-create-2-factory
          Use the create 2 factory in all cases including tests and non-
broadcasting scripts

  -v, --verbosity...
          Verbosity of the EVM.

          Pass multiple times to increase the verbosity (e.g. -v, -vv, -vvv).

          Verbosity levels:
          - 2: Print logs for all tests
          - 3: Print execution traces for failing tests
          - 4: Print execution traces for all tests, and setup traces for
failing tests
          - 5: Print execution and setup traces for all tests

Fork config:
      --compute-units-per-second <CUPS>
          Sets the number of assumed available compute units per second for
this provider

          default value: 330

          See also --fork-url and
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

      --no-rpc-rate-limit
          Disables rate limiting for this node's provider.

          See also --fork-url and
          https://docs.alchemy.com/reference/compute-units#what-are-cups-
compute-units-per-second

          [aliases: no-rate-limit]

Executor environment config:
      --gas-limit <GAS_LIMIT>
          The block gas limit
```

```
      --code-size-limit <CODE_SIZE>
          EIP-170: Contract code size limit in bytes. Useful to increase this
because of tests. By
          default, it is 0x6000 (~25kb)

      --chain <CHAIN>
          The chain name or EIP-155 chain ID

          [aliases: chain-id]

      --gas-price <GAS_PRICE>
          The gas price

      --block-base-fee-per-gas <FEE>
          The base fee in a block

          [aliases: base-fee]

      --tx-origin <ADDRESS>
          The transaction origin

      --block-coinbase <ADDRESS>
          The coinbase of the block

      --block-timestamp <TIMESTAMP>
          The timestamp of the block

      --block-number <BLOCK>
          The block number

      --block-difficulty <DIFFICULTY>
          The block difficulty

      --block-prevrandao <PREVRANDAO>
          The block prevrandao value. NOTE: Before merge this field was
mix_hash

      --block-gas-limit <GAS_LIMIT>
          The block gas limit

      --memory-limit <MEMORY_LIMIT>
          The memory limit per EVM execution in bytes. If this limit is
exceeded, a `MemoryLimitOOG`
          result is thrown.

          The default is 128MiB.

      --disable-block-gas-limit
          Whether to disable the block gas limit checks

          [aliases: no-gas-limit]

      --isolate
          Whether to enable isolation of calls. In isolation mode all top-level
calls are executed
          as a separate transaction in a separate EVM context, enabling more
```

```
precise gas accounting
          and transaction state changes
```

chisel list

List all cached sessions

```
$ chisel list --help
Usage: chisel list

Options:
  -h, --help  Print help
```

chisel load

Load a cached session

```
$ chisel load --help
Usage: chisel load <ID>

Arguments:
  <ID>  The ID of the session to load

Options:
  -h, --help  Print help
```

chisel view

View the source of a cached session

```
$ chisel view --help
Usage: chisel view <ID>

Arguments:
  <ID>  The ID of the session to load

Options:
  -h, --help  Print help
```

chisel clear-cache

Clear all cached chisel sessions from the cache directory

```
$ chisel clear-cache --help
Usage: chisel clear-cache

Options:
  -h, --help  Print help
```

forge Commands

- General Commands
- Project Commands
- Build Commands
- Test Commands
- Deploy Commands
- Utility Commands

General Commands

- forge
- forge help
- forge completions

forge

NAME

forge - Build, test, fuzz, debug and deploy Solidity contracts.

SYNOPSIS

`forge` [*options*] *command* [*args*]
`forge` [*options*] `--version`
`forge` [*options*] `--help`

DESCRIPTION

This program is a set of tools to build, test, fuzz, debug and deploy Solidity smart contracts.

COMMANDS

General Commands

forge help
: Display help information about Forge.

forge completions
: Generate shell autocompletions for Forge.

Project Commands

forge init
: Create a new Forge project.

forge install
: Install one or multiple dependencies.

forge update
: Update one or multiple dependencies.

forge remove
Remove one or multiple dependencies.

forge config
Display the current config.

forge remappings
Get the automatically inferred remappings for this project.

forge tree
Display a tree visualization of the project's dependency graph.

forge geiger Detects usage of unsafe cheat codes in a foundry project and its dependencies.

Build Commands

forge build
Build the project's smart contracts.

forge clean
Remove the build artifacts and cache directories.

forge inspect
Get specialized information about a smart contract.

Test Commands

forge test
Run the project's tests.

forge snapshot
Create a snapshot of each test's gas usage.

forge coverage
Generate coverage reports.

Deploy Commands

forge create
Deploy a smart contract.

forge verify-contract
Verify smart contracts on Etherscan.

forge verify-check
Check verification status on Etherscan.

forge flatten

Flatten a source file and all of its imports into one file.

Utility Commands

forge debug

Debug a single smart contract as a script.

forge bind

Generate Rust bindings for smart contracts.

forge cache

Manage the Foundry cache.

forge cache clean

Cleans cached data from `~/.foundry`.

forge cache ls

Shows cached data from `~/.foundry`.

forge script

Run a smart contract as a script, building transactions that can be sent onchain.

forge upload-selectors

Uploads abi of given contract to https://sig.eth.samczsun.com function selector database.

forge doc

Generate documentation for Solidity source files.

OPTIONS

Special Options

`-V`

`--version`

Print version info and exit.

Common Options

`-h`

`--help`

Prints help information.

FILES

`~/.foundry/`
Default location for Foundry's "home" directory where it stores various files.

`~/.foundry/bin/`
Binaries installed using `foundryup` will be located here.

`~/.foundry/cache/`
Forge's cache directory, where it stores cached block data and more.

`~/.foundry/foundry.toml`
The global Foundry config.

`~/.svm`
The location of the Forge-managed solc binaries.

EXAMPLES

1. Create a new Forge project:

```
forge init hello_foundry
```

2. Build a project:

```
forge build
```

3. Run a project's tests:

```
forge test
```

BUGS

See https://github.com/foundry-rs/foundry/issues for issues.

forge help

NAME

forge-help - Get help for a Forge command

SYNOPSIS

`forge help` [*subcommand*]

DESCRIPTION

Prints a help message for the given command.

EXAMPLES

1. Get help for a command:

   ```
   forge help build
   ```

2. Help is also available with the `--help` flag:

   ```
   forge build --help
   ```

SEE ALSO

forge

forge completions

NAME

forge-completions - Generate shell completions script

SYNOPSIS

`forge completions` *shell*

DESCRIPTION

Generates a shell completions script for the given shell.

Supported shells are:

- bash
- elvish
- fish
- powershell
- zsh

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Generate shell completions script for zsh:

```
forge completions zsh > $HOME/.oh-my-zsh/completions/_forge
```

SEE ALSO

forge

Project Commands

- forge init
- forge install
- forge update
- forge remove
- forge config
- forge remappings
- forge tree
- forge geiger

forge init

NAME

forge-init - Create a new Forge project.

SYNOPSIS

`forge init` [*options*] [*root*]

DESCRIPTION

Create a new Forge project in the directory *root* (by default the current working directory).

The default template creates the following project layout:

```
.
├── README.md
├── foundry.toml
├── lib
│   └── forge-std
│       ├── LICENSE-APACHE
│       ├── LICENSE-MIT
│       ├── README.md
│       ├── foundry.toml
│       ├── package.json
│       ├── scripts
│       ├── src
│       └── test
├── script
│   └── Counter.s.sol
├── src
│   └── Counter.sol
└── test
    └── Counter.t.sol

8 directories, 10 files
```

However, it is possible to create a project from another using `--template`.

By default, `forge init` will also initialize a new git repository, install some submodules and create an initial commit message.

If you do not want this behavior, pass `--no-git`.

OPTIONS

Init Options

`--force`
Create the project even if the specified root directory is not empty.

`-t` *template*
`--template` *template*
The template to start from.

`--vscode`
Create a `.vscode/settings.json` file with Solidity settings, and generate a `remappings.txt` file.

`--offline`
Do not install dependencies from the network.

VCS Options

`--no-commit`
Do not create an initial commit.

`--no-git`
Do not create a git repository.

Display Options

`-q`
`--quiet`
Do not print any messages.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Create a new project:

```
forge init hello_foundry
```

2. Create a new project, but do not create a git repository:

```
forge init --no-git hello_foundry
```

3. Forcibly create a new project in a non-empty directory:

```
forge init --force
```

SEE ALSO

forge

forge install

NAME

forge-install - Install one or more dependencies.

SYNOPSIS

`forge install` [*options*] [*deps...*]

DESCRIPTION

Install one or more dependencies.

Dependencies are installed as git submodules. If you do not want this behavior, pass `--no-git`.

If no arguments are provided, then existing dependencies are installed.

Dependencies can be a raw URL (`https://foo.com/dep`), an SSH URL (`git@github.com:owner/repo`), or the path to a GitHub repository (`owner/repo`). Additionally, a ref can be added to the dependency path to install a specific version of a dependency.

A ref can be:

- A branch: `owner/repo@master`
- A tag: `owner/repo@v1.2.3`
- A commit: `owner/repo@8e8128`

The ref defaults to `master`.

You can also choose the name of the folder the dependency will be in. By default, the folder name is the name of the repository. If you want to change the name of the folder, prepend `<folder>=` to the dependency.

OPTIONS

Project Options

`--root` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

VCS Options

`--no-commit`
Do not create a commit.

`--no-git`
Install without adding the dependency as a submodule.

Display Options

`-q`
`--quiet`
Do not print any messages.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Install a dependency:

```
forge install transmissions11/solmate
```

2. Install a specific version of a dependency:

```
forge install transmissions11/solmate@v7
```

3. Install multiple dependencies:

```
forge install transmissions11/solmate@v7 OpenZeppelin/openzeppelin-contracts
```

4. Install a dependency without creating a submodule:

```
forge install --no-git transmissions11/solmate
```

5. Install a dependency in a specific folder:

```
forge install soulmate=transmissions11/solmate
```

SEE ALSO

forge, forge update, forge remove

forge update

NAME

forge-update - Update one or more dependencies.

SYNOPSIS

`forge update` [*options*] [*dep*]

DESCRIPTION

Update one or more dependencies.

The argument *dep* is a path to the dependency you want to update. Forge will update to the latest version on the ref you specified for the dependency when you ran `forge install`.

If no argument is provided, then all dependencies are updated.

OPTIONS

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Update a dependency:

```
forge update lib/solmate
```

2. Update all dependencies:

```
forge update
```

SEE ALSO

forge, forge install, forge remove

forge remove

NAME

forge-remove - Remove one or multiple dependencies.

SYNOPSIS

`forge remove` [*options*] [*deps...*]

DESCRIPTION

Remove one or multiple dependencies.

Dependencies can be a raw URL (`https://foo.com/dep`), the path to a GitHub repository (`owner/repo`) or the path to the dependency in the project tree.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Remove a dependency by path:

```
forge remove lib/solmate
```

2. Remove a dependency by GitHub repository name:

```
forge remove dapphub/solmate
```

SEE ALSO

forge, forge install, forge update

forge config

NAME

forge-config - Display the current config.

SYNOPSIS

`forge config` [*options*]

DESCRIPTION

Display the current config.

This command can be used to create a new basic `foundry.toml` or to see what values are currently set, taking environment variables and the global configuration file into account.

The command supports almost all flags of the other commands in Forge to allow overriding values in the displayed configuration.

OPTIONS

Config Options

`--basic`
: Prints a basic configuration file.

`--fix`
: Attempts to fix any configuration warnings.

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Create a new basic config:

```
forge config > foundry.toml
```

2. Enable FFI in `foundry.toml`:

```
forge config --ffi > foundry.toml
```

SEE ALSO

forge

forge remappings

NAME

forge-remappings - Get the automatically inferred remappings for the project.

SYNOPSIS

`forge remappings` [*options*]

DESCRIPTION

Get the automatically inferred remappings for the project.

OPTIONS

Project Options

`--root` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`--lib-path` *path*
The path to the library folder.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Create a `remappings.txt` file from the inferred remappings:

```
forge remappings > remappings.txt
```

SEE ALSO

forge

forge tree

NAME

forge-tree - Display a tree visualization of the project's dependency graph.

SYNOPSIS

`forge tree` [*options*]

DESCRIPTION

Display a visualization of the project's dependency graph.

```
$ forge tree
src/OpenZeppelinNft.sol 0.8.10
├── lib/openzeppelin-contracts/contracts/token/ERC721/ERC721.sol ^0.8.0
│   ├── lib/openzeppelin-contracts/contracts/token/ERC721/IERC721.sol ^0.8.0
│   │   └── lib/openzeppelin-
contracts/contracts/utils/introspection/IERC165.sol ^0.8.0
│   ├── lib/openzeppelin-contracts/contracts/token/ERC721/IERC721Receiver.sol
^0.8.0
│   ├── lib/openzeppelin-
contracts/contracts/token/ERC721/extensions/IERC721Metadata.sol ^0.8.0
│   │   └── lib/openzeppelin-contracts/contracts/token/ERC721/IERC721.sol
^0.8.0 (*)
│   ├── lib/openzeppelin-contracts/contracts/utils/Address.sol ^0.8.1
│   ├── lib/openzeppelin-contracts/contracts/utils/Context.sol ^0.8.0
│   ├── lib/openzeppelin-contracts/contracts/utils/Strings.sol ^0.8.0
│   └── lib/openzeppelin-contracts/contracts/utils/introspection/ERC165.sol
^0.8.0
│       └── lib/openzeppelin-
contracts/contracts/utils/introspection/IERC165.sol ^0.8.0
├── lib/openzeppelin-contracts/contracts/utils/Strings.sol ^0.8.0
├── lib/openzeppelin-contracts/contracts/security/PullPayment.sol ^0.8.0
│   └── lib/openzeppelin-contracts/contracts/utils/escrow/Escrow.sol ^0.8.0
│       ├── lib/openzeppelin-contracts/contracts/access/Ownable.sol ^0.8.0
│       │   └── lib/openzeppelin-contracts/contracts/utils/Context.sol ^0.8.0
│       └── lib/openzeppelin-contracts/contracts/utils/Address.sol ^0.8.1
└── lib/openzeppelin-contracts/contracts/access/Ownable.sol ^0.8.0 (*)
src/SolmateNft.sol 0.8.10
├── lib/solmate/src/tokens/ERC721.sol >=0.8.0
├── lib/openzeppelin-contracts/contracts/utils/Strings.sol ^0.8.0
├── lib/openzeppelin-contracts/contracts/security/PullPayment.sol ^0.8.0 (*)
└── lib/openzeppelin-contracts/contracts/access/Ownable.sol ^0.8.0 (*)
test/OpenZeppelinNft.t.sol 0.8.10
├── lib/forge-std/src/Test.sol >=0.6.0 <0.9.0
│   ├── lib/forge-std/src/Script.sol >=0.6.0 <0.9.0
│   │   ├── lib/forge-std/src/Vm.sol >=0.6.0
│   │   ├── lib/forge-std/src/console.sol >=0.4.22 <0.9.0
│   │   └── lib/forge-std/src/console2.sol >=0.4.22 <0.9.0
│   └── lib/solmate/lib/ds-test/src/test.sol >=0.4.23
├── src/OpenZeppelinNft.sol 0.8.10 (*)
└── lib/openzeppelin-contracts/contracts/token/ERC721/IERC721Receiver.sol
^0.8.0
test/SolmateNft.sol 0.8.10
├── lib/forge-std/src/Test.sol >=0.6.0 <0.9.0 (*)
└── src/SolmateNft.sol 0.8.10 (*)
```

OPTIONS

Flatten Options

`--charset` *charset*

Character set to use in output: utf8, ascii. Default: utf8

`--no-dedupe`
Do not de-duplicate (repeats all shared dependencies)

Project Options

`--build-info`
Generate build info files.

`--build-info-path` *path*
Output path to directory that build info files will be written to.

`--root` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`-C` *path*
`--contracts` *path*
The contracts source directory.
Environment: `DAPP_SRC`

`--lib-paths` *path*
The path to the library folder.

`-R` *remappings*
`--remappings` *remappings*
The project's remappings.

The parameter is a comma-separated list of remappings in the format `<source>=<dest>`.

`--cache-path` *path*
The path to the compiler cache.

`--config-path` *file*
Path to the config file.

`--hh`
`--hardhat`
This is a convenience flag, and is the same as passing `--contracts contracts --lib-paths node-modules`.

Common Options

`-h`
`--help`
Prints help information.

SEE ALSO

forge

forge geiger

NAME

forge-geiger - Detects usage of unsafe cheat codes in a foundry project and its dependencies.

SYNOPSIS

`forge geiger` [*options*] [*path*]

DESCRIPTION

Detects usage of unsafe cheat codes in a foundry project and its dependencies.

OPTIONS

`--root` *path*
: The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`--check`
: Run in 'check' mode. Exits with 0 if no unsafe cheat codes were found. Exits with 1 if unsafe cheat codes are detected.

`--full`
: Print a full report of all files even if no unsafe functions are found.

Common Options

`-h`
`--help`
: Prints help information.

SEE ALSO

forge

Build Commands

- forge build
- forge clean
- forge inspect

forge build

NAME

forge-build - Build the project's smart contracts.

SYNOPSIS

`forge build` or `forge b` [*options*]

DESCRIPTION

Build the project's smart contracts.

The command will try to detect the latest version that can compile your project by looking at the version requirements of all your contracts and dependencies.

You can override this behaviour by passing `--no-auto-detect`. Alternatively, you can pass `--use <SOLC_VERSION>`.

If the command detects that the Solidity compiler version it is using to build is not installed, it will download it and install it in `~/.svm`. You can disable this behavior by passing `--offline`.

The build is incremental, and the build cache is saved in `cache/` in the project root by default. If you want to clear the cache, pass `--force`, and if you want to change the cache directory, pass `--cache-path <PATH>`.

Build Modes

There are three build modes:

- Just compilation (default): Builds the project and saves the contract artifacts in `out/` (or the path specified by `--out <PATH>`).
- Size mode (`--sizes`): Builds the project, displays the size of non-test contracts and exits with code 1 if any of them are above the size limit.
- Name mode (`--names`): Builds the project, displays the names of the contracts and exits.

The Optimizer

You can enable the optimizer by passing `--optimize`, and you can adjust the number of optimizer runs by passing `--optimizer-runs <RUNS>`.

You can also opt-in to the Solidity IR compilation pipeline by passing `--via-ir`. Read more about the IR pipeline in the Solidity docs.

By default, the optimizer is enabled and runs for 200 cycles.

Conditional Optimizer Usage

Many projects use the solc optimizer, either through the standard compilation pipeline or the IR pipeline. But in some cases, the optimizer can significantly slow down compilation speeds.

A config file for a project using the optimizer may look like this for regular compilation:

```
[profile.default]
solc-version = "0.8.17"
optimizer = true
optimizer-runs = 10_000_000
```

Or like this for `via-ir`:

```
[profile.default]
solc-version = "0.8.17"
via_ir = true
```

To reduce compilation speeds during development and testing, one approach is to have a `lite` profile that has the optimizer off and use this for development/testing cycle. The updated config file for regular compilation may look like this:

```
[profile.default]
solc-version = "0.8.17"
optimizer = true
optimizer-runs = 10_000_000

[profile.lite]
optimizer = false
```

Or like this for `via-ir`:

```
[profile.default]
solc-version = "0.8.17"
via_ir = true

[profile.lite.optimizer_details.yulDetails]
optimizerSteps = ''
```

When setup like this, `forge build` (or `forge test` / `forge script`) still uses the standard profile, so by default a `forge script` invocation will deploy your contracts with the production setting. Running `FOUNDRY_PROFILE=lite forge build` (and again, same for the test and script commands) will use the lite profile to reduce compilation times.

> There are additional optimizer details you can configure, see the Additional Optimizer Settings section below for more info.

Artifacts

You can add extra output from the Solidity compiler to your artifacts by passing `--extra-output <SELECTOR>`.

The selector is a path in the Solidity compiler output, which you can read more about in the Solidity docs.

You can also write some of the compiler output to separate files by passing `--extra-output-files <SELECTOR>`.

Valid selectors for `--extra-output-files` are:

- `metadata`: Written as a `metadata.json` file in the artifacts directory
- `ir`: Written as a `.ir` file in the artifacts directory
- `irOptimized`: Written as a `.iropt` file in the artifacts directory
- `ewasm`: Written as a `.ewasm` file in the artifacts directory
- `evm.assembly`: Written as a `.asm` file in the artifacts directory

Watch Mode

The command can be run in watch mode by passing `--watch [PATH...]`, which will rebuild every time a watched file or directory is changed. The source directory is watched by default.

Sparse Mode (experimental)

Sparse mode only compiles files that match certain criteria.

By default, this filter applies to files that have not been changed since the last build, but for commands that take file filters (e.g. forge test), sparse mode will only recompile files that match the filter.

Sparse mode is opt-in until the feature is stabilized. To opt-in to sparse mode and try it out, set `sparse_mode` in your configuration file.

Additional Optimizer Settings

The optimizer can be fine-tuned with additional settings. Simply set the `optimizer_details` table in your configuration file. For example:

```
[profile.default.optimizer_details]
constantOptimizer = true
yul = true

[profile.default.optimizer_details.yulDetails]
stackAllocation = true
optimizerSteps = 'dhfoDgvulfnTUtnIf'
```

See the compiler input description documentation for more information on available settings (specifically `settings.optimizer.details`).

Revert Strings

You can control how revert strings are generated by the compiler. By default, only user supplied revert strings are included in the bytecode, but there are other options:

- `strip` : Removes all revert strings (if possible, i.e. if literals are used) keeping side-effects.
- `debug` : Injects strings for compiler-generated internal reverts, implemented for ABI encoders V1 and V2 for now.
- `verboseDebug` : Appends further information to user-supplied revert strings (not yet implemented).

Additional Model Checker settings

Solidity's built-in model checker is an opt-in module that can be enabled via the `ModelChecker` object.

See Compiler Input Description `settings.modelChecker` and the model checker's options.

The module is available in `solc` release binaries for OSX and Linux. The latter requires the z3 library version [4.8.8, 4.8.14] to be installed in the system (SO version 4.8).

Similarly to the optimizer settings above, the `model_checker` settings must be prefixed with the profile they correspond to: `[profile.default.model_checker]` belongs to the `[profile.default]` .

```
## foundry.toml
[profile.default.model_checker]
contracts = { '/path/to/project/src/Contract.sol' = [ 'Contract' ] }
engine = 'chc'
timeout = 10000
targets = [ 'assert' ]
```

The fields above are recommended when using the model checker. Setting which contract should be verified is extremely important, otherwise all available contracts will be verified which can consume a lot of time. The recommended engine is `chc`, but `bmc` and `all` (runs both) are also accepted. It is also important to set a proper timeout (given in milliseconds), since the default time given to the underlying solvers may not be enough. If no verification targets are given, only assertions will be checked.

The model checker will run when `forge build` is invoked, and will show findings as warnings if any.

OPTIONS

Build Options

`--names` Print compiled contract names.

`--sizes` Print compiled non-test contract sizes, exiting with code 1 if any of them are above the size limit.

`--skip` Skip compilation of non-essential contract directories like test or script (usage `--skip test`).

Cache Options

`--force`
Clear the cache and artifacts folder and recompile.

Linker Options

`--libraries` *libraries*
Set pre-linked libraries.

The parameter must be in the format `<remapped path to lib>:<library name>:<address>`, e.g. `src/Contract.sol:Library:0x....`

Can also be set in your configuration file as `libraries = ["<path>:<lib name>:<address>"]`.

Compiler Options

`--optimize`
Activate the Solidity optimizer.

`--optimizer-runs` *runs*
The number of optimizer runs.

`--via-ir`
Use the Yul intermediate representation compilation pipeline.

`--revert-strings`
How to treat revert and require reason strings.

`--use` *solc_version*
Specify the solc version, or a path to a local solc, to build with.

Valid values are in the format `x.y.z`, `solc:x.y.z` or `path/to/solc`.

`--offline`
Do not access the network. Missing solc versions will not be installed.

`--no-auto-detect`
Do not auto-detect solc.

`--ignored-error-codes` *error_codes*
Ignore solc warnings by error code. The parameter is a comma-separated list of error codes.

`--extra-output` *selector*
Extra output to include in the contract's artifact.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--extra-output-files` *selector*
Extra output to write to separate files.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--evm-version` *version*
The target EVM version.

Project Options

`--build-info`
Generate build info files.

`--build-info-path` *path*
Output path to directory that build info files will be written to.

`--root` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`-C` *path*
`--contracts` *path*
The contracts source directory.
Environment: `DAPP_SRC`

`--lib-paths` *path*
The path to the library folder.

`-R` *remappings*
`--remappings` *remappings*
The project's remappings.

The parameter is a comma-separated list of remappings in the format `<source>=<dest>`.

`--cache-path` *path*
The path to the compiler cache.

`--config-path` *file*
Path to the config file.

`--hh`
`--hardhat`
This is a convenience flag, and is the same as passing `--contracts contracts --lib-paths node-modules`.

`-o` *path*
`--out` *path*
The project's artifacts directory.

`--silent`
Suppress all output.

Watch Options

`-w` [*path...*]
`--watch` [*path...*]
Watch specific file(s) or folder(s).

By default, the project's source directory is watched.

`-d` *delay*
`--delay` *delay*
File update debounce delay.

During the delay, incoming change events are accumulated and only once the delay has passed, is an action taken.
Note that this does not mean a command will be started: if `--no-restart` is given and a command is already running, the outcome of the action will be to do nothing.

Defaults to 50ms. Parses as decimal seconds by default, but using an integer with the `ms` suffix may be more convenient.

When using `--poll` mode, you'll want a larger duration, or risk overloading disk I/O.

`--no-restart`
Do not restart the command while it's running.

`--run-all`
Explicitly re-run the command on all files when a change is made.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Build the project:

```
forge build
```

2. Build the project with solc 0.6.0:

```
forge build --use solc:0.6.0
```

3. Build the project with additional artifact output:

```
forge build --extra-output evm.assembly
```

4. Build the project in watch mode:

```
forge build --watch
```

SEE ALSO

forge, forge clean, forge inspect

forge clean

NAME

forge-clean - Remove the build artifacts and cache directories.

SYNOPSIS

`forge clean` [*options*]

DESCRIPTION

Remove the build artifacts and cache directories.

OPTIONS

Clean Options

`--root` *path*
: The project's root path. Defaults to the current working directory.

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Clean artifacts and cache in a project:

```
forge clean
```

SEE ALSO

forge

forge inspect

NAME

forge-inspect - Get specialized information about a smart contract

SYNOPSIS

`forge inspect` [*options*] *contract_name field*

DESCRIPTION

Get specialized information about a smart contract.

The field to inspect (*field*) can be any of:

- `abi`
- `b` / `bytes` / `bytecode`
- `deployedBytecode` / `deployed_bytecode` / `deployed-bytecode` / `deployedbytecode` / `deployed`
- `assembly` / `asm`
- `asmOptimized` / `assemblyOptimized` / `assemblyoptimized` / `assembly_optimized` / `asmopt` / `assembly-optimized` / `asmo` / `asm-optimized` / `asmoptimized` / `asm_optimized`
- `methods` / `methodidentifiers` / `methodIdentifiers` / `method_identifiers` / `method-identifiers` / `mi`
- `gasEstimates` / `gas` / `gas_estimates` / `gas-estimates` / `gasestimates`
- `storageLayout` / `storage_layout` / `storage-layout` / `storagelayout` / `storage`
- `devdoc` / `dev-doc` / `devDoc`
- `ir`
- `ir-optimized` / `irOptimized` / `iroptimized` / `iro` / `iropt`
- `metadata` / `meta`
- `userdoc` / `userDoc` / `user-doc`
- `ewasm` / `e-wasm`

OPTIONS

`--pretty`
: Pretty print the selected field, if supported.

Cache Options

`--force`
Clear the cache and artifacts folder and recompile.

Linker Options

`--libraries` *libraries*
Set pre-linked libraries.

The parameter must be in the format `<remapped path to lib>:<library name>:<address>`, e.g. `src/Contract.sol:Library:0x...`.

Can also be set in your configuration file as `libraries = ["<path>:<lib name>:<address>"]`.

Compiler Options

`--optimize`
Activate the Solidity optimizer.

`--optimizer-runs` *runs*
The number of optimizer runs.

`--via-ir`
Use the Yul intermediate representation compilation pipeline.

`--revert-strings`
How to treat revert and require reason strings.

`--use` *solc_version*
Specify the solc version, or a path to a local solc, to build with.

Valid values are in the format `x.y.z`, `solc:x.y.z` or `path/to/solc`.

`--offline`
Do not access the network. Missing solc versions will not be installed.

`--no-auto-detect`
Do not auto-detect solc.

`--ignored-error-codes` *error_codes*
Ignore solc warnings by error code. The parameter is a comma-separated list of error codes.

`--extra-output` *selector*
Extra output to include in the contract's artifact.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--extra-output-files` *selector*
Extra output to write to separate files.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--evm-version` *version*
The target EVM version.

Project Options

`--build-info`
Generate build info files.

`--build-info-path` *path*
Output path to directory that build info files will be written to.

`--root` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`-C` *path*
`--contracts` *path*
The contracts source directory.
Environment: `DAPP_SRC`

`--lib-paths` *path*
The path to the library folder.

`-R` *remappings*
`--remappings` *remappings*
The project's remappings.

The parameter is a comma-separated list of remappings in the format `<source>=<dest>`.

`--cache-path` *path*
The path to the compiler cache.

`--config-path` *file*
Path to the config file.

`--hh`
`--hardhat`
This is a convenience flag, and is the same as passing `--contracts contracts --lib-paths node-modules`.

`-o` *path*
`--out` *path*
The project's artifacts directory.

`--silent`
Suppress all output.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Inspect the bytecode of a contract:

```
forge inspect MyContract bytecode
```

2. Inspect the storage layout of a contract:

```
forge inspect MyContract storage
```

3. Inspect the abi of a contract in a pretty format:

```
forge inspect --pretty MyContract abi
```

SEE ALSO

forge, forge build

Test Commands

- forge test
- forge snapshot
- forge coverage

forge test

NAME

forge-test - Run the project's tests.

SYNOPSIS

`forge test` [*options*]

DESCRIPTION

Run the project's tests.

Forking

It is possible to run the tests in a forked environment by passing `--fork-url <URL>`.

When the tests are running in a forked environment, you can access all the state of the forked chain as you would if you had deployed the contracts. Cheatcodes are still available.

You can also specify a block number to fork from by passing `--fork-block-number <BLOCK>`. When forking from a specific block, the chain data is cached to `~/.foundry/cache`. If you do not want to cache the chain data, pass `--no-storage-caching`.

Traces that cannot be decoded by local contracts when running in a forked environment (e.g. calls to contracts that live on mainnet, like tokens) can optionally be decoded using Etherscan. To use Etherscan for trace decoding, set `ETHERSCAN_API_KEY` or pass `--etherscan-api-key <KEY>`.

Debugging

It is possible to run a test in an interactive debugger. To start the debugger, pass `--debug <TEST>`.

If multiple tests match the specified pattern, you must use other test filters in order to reduce the matching number of tests to exactly 1.

If the test is a unit test, it is immediately opened in the debugger.

If the test is a fuzz test, the fuzz test is run and the debugger is opened on the first failing scenario. If there are no failing scenarios for the fuzz test, the debugger is opened on the last scenario.

More information on the debugger can be found in the debugger chapter.

Gas reports

You can generate a gas report by passing `--gas-report`.

More information on gas reports can be found in the gas reports chapter.

List

It is possible to list the tests without running them. You can pass `--json` to make it easier for outside extensions to parse structured content.

OPTIONS

Test Options

`-m` *regex*
`--match` *regex*
: Only run test functions matching the specified regex pattern.
Deprecated: See `--match-test`.

`--match-test` *regex*
: Only run test functions matching the specified regex pattern.

`--no-match-test` *regex*
: Only run test functions that do not match the specified regex pattern.

`--match-contract` *regex*
: Only run tests in contracts matching the specified regex pattern.

`--no-match-contract` *regex*
: Only run tests in contracts that do not match the specified regex pattern.

`--match-path` *glob*
: Only run tests in source files matching the specified glob pattern.

`--no-match-path` *glob*
: Only run tests in source files that do not match the specified glob pattern.

`--debug` *regex*
Run a test in the debugger.

The argument passed to this flag is the name of the test function you want to run, and it works the same as `--match-test`.

If more than one test matches your specified criteria, you must add additional filters until only one test is found (see `--match-contract` and `--match-path`).

The matching test will be opened in the debugger regardless of the outcome of the test.

If the matching test is a fuzz test, then it will open the debugger on the first failure case. If the fuzz test does not fail, it will open the debugger on the last fuzz case.

For more fine-grained control of which fuzz case is run, see `forge debug`.

`--gas-report`
Print a gas report.

`--allow-failure`
Exit with code 0 even if a test fails.

`--fail-fast`
Stop running tests after the first failure.

`--etherscan-api-key` *key*
Etherscan API key. If set, traces are decoded using Etherscan if `--fork-url` is also set.
Environment: `ETHERSCAN_API_KEY`

EVM Options

`-f` *url*
`--rpc-url` *url*
`--fork-url` *url*
Fetch state over a remote endpoint instead of starting from an empty state.

If you want to fetch state from a specific block number, see `--fork-block-number`.

`--fork-block-number` *block*
Fetch state from a specific block number over a remote endpoint. See `--fork-url`.

`--fork-retry-backoff <BACKOFF>`
Initial retry backoff on encountering errors.

`--no-storage-caching`
Explicitly disables the use of RPC caching.

All storage slots are read entirely from the endpoint. See `--fork-url`.

`-v`
`--verbosity`
Verbosity of the EVM.

Pass multiple times to increase the verbosity (e.g. `-v`, `-vv`, `-vvv`).

Verbosity levels:
- 2: Print logs for all tests
- 3: Print execution traces for failing tests
- 4: Print execution traces for all tests, and setup traces for failing tests
- 5: Print execution and setup traces for all tests

`--sender` *address*
The address which will be executing tests

`--initial-balance` *balance*
The initial balance of deployed contracts

`--ffi`
Enables the FFI cheatcode

Executor Options

`--base-fee <FEE>`
`--block-base-fee-per-gas <FEE>`
The base fee in a block (in wei).

`--block-coinbase` *address*
The coinbase of the block.

`--block-difficulty` *difficulty*
The block difficulty.

`--block-gas-limit` *gas_limit*
The block gas limit.

`--block-number` *block*
The block number.

`--block-timestamp` *timestamp*
The timestamp of the block (in seconds).

`--chain-id` *chain_id*
The chain ID.

`--gas-limit` *gas_limit*
The block gas limit.

`--gas-price` *gas_price*
The gas price (in wei).

`--tx-origin` *address*
The transaction origin.

Cache Options

`--force`
Clear the cache and artifacts folder and recompile.

Linker Options

`--libraries` *libraries*
Set pre-linked libraries.

The parameter must be in the format `<remapped path to lib>:<library name>:<address>`, e.g. `src/Contract.sol:Library:0x...`.

Can also be set in your configuration file as `libraries = ["<path>:<lib name>:<address>"]`.

Compiler Options

`--optimize`
Activate the Solidity optimizer.

`--optimizer-runs` *runs*
The number of optimizer runs.

`--via-ir`
Use the Yul intermediate representation compilation pipeline.

`--revert-strings`
How to treat revert and require reason strings.

`--use` *solc_version*
Specify the solc version, or a path to a local solc, to build with.

Valid values are in the format `x.y.z`, `solc:x.y.z` or `path/to/solc`.

`--offline`
Do not access the network. Missing solc versions will not be installed.

`--no-auto-detect`
Do not auto-detect solc.

`--ignored-error-codes` *error_codes*
Ignore solc warnings by error code. The parameter is a comma-separated list of error codes.

`--extra-output` *selector*
Extra output to include in the contract's artifact.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--extra-output-files` *selector*
Extra output to write to separate files.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--evm-version` *version*
The target EVM version.

Project Options

`--build-info`
Generate build info files.

`--build-info-path` *path*
Output path to directory that build info files will be written to.

`--root` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`-C` *path*
`--contracts` *path*
The contracts source directory.
Environment: `DAPP_SRC`

`--lib-paths` *path*
The path to the library folder.

`-R` *remappings*
`--remappings` *remappings*
The project's remappings.

The parameter is a comma-separated list of remappings in the format `<source>=<dest>`.

`--cache-path` *path*
The path to the compiler cache.

`--config-path` *file*
Path to the config file.

`--hh`
`--hardhat`
This is a convenience flag, and is the same as passing `--contracts contracts --lib-paths node-modules`.

`-o` *path*
`--out` *path*
The project's artifacts directory.

`--silent`
Suppress all output.

Watch Options

`-w` [*path...*]
`--watch` [*path...*]
Watch specific file(s) or folder(s).

By default, the project's source directory is watched.

`-d` *delay*
`--delay` *delay*
File update debounce delay.

During the delay, incoming change events are accumulated and only once the delay has passed, is an action taken.
Note that this does not mean a command will be started: if `--no-restart` is given and a command is already running, the outcome of the action will be to do nothing.

Defaults to 50ms. Parses as decimal seconds by default, but using an integer with the `ms` suffix may be more convenient.

When using `--poll` mode, you'll want a larger duration, or risk overloading disk I/O.

`--no-restart`
Do not restart the command while it's running.

`--run-all`
Explicitly re-run the command on all files when a change is made.

Display Options

`-j`
`--json`
Print the deployment information as JSON.

`--list`
List tests instead of running them.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Run the tests:

```
forge test
```

2. Open a test in the debugger:

```
forge test --debug testSomething
```

3. Generate a gas report:

```
forge test --gas-report
```

4. Only run tests in `test/Contract.t.sol` in the `BigTest` contract that start with `testFail`:

```
forge test --match-path test/Contract.t.sol --match-contract BigTest \
  --match-test "testFail*"
```

5. List tests in desired format

```
forge test --list
forge test --list --json
forge test --list --json --match-test "testFail*" | tail -n 1 | json_pp
```

SEE ALSO

forge, forge build, forge snapshot

forge snapshot

NAME

forge-snapshot - Create a snapshot of each test's gas usage.

SYNOPSIS

`forge snapshot` [*options*]

DESCRIPTION

Create a snapshot of each test's gas usage.

The results are written to a file named `.gas-snapshot`. You can change the name of the file by passing `--snap <PATH>`.

Fuzz tests are included by default in the snapshot. They use a static seed to achieve deterministic results.

Snapshots can be compared with `--diff` and `--check`. The first flag will output a diff, and the second will output a diff *and* exit with code 1 if the snapshots do not match.

OPTIONS

Snapshot Options

`--asc`
Sort results by gas used (ascending).

`--desc`
Sort results by gas used (descending).

`--min` *min_gas*
Only include tests that used more gas that the given amount.

`--max` *max_gas*
Only include tests that used less gas that the given amount.

`--tolerance` *threshold*
Tolerates gas deviations up to the specified percentage (0-100).

`--diff` *path*
Output a diff against a pre-existing snapshot.

By default the comparison is done with `.gas-snapshot`.

`--check` *path*
Compare against a pre-existing snapshot, exiting with code 1 if they do not match.

Outputs a diff if the snapshots do not match.

By default the comparison is done with `.gas-snapshot`.

`--snap` *path*
Output file for the snapshot. Default: `.gas-snapshot`.

Test Options

`-m` *regex*
`--match` *regex*
Only run test functions matching the specified regex pattern.
Deprecated: See `--match-test`.

`--match-test` *regex*
Only run test functions matching the specified regex pattern.

`--no-match-test` *regex*
Only run test functions that do not match the specified regex pattern.

`--match-contract` *regex*
Only run tests in contracts matching the specified regex pattern.

`--no-match-contract` *regex*
Only run tests in contracts that do not match the specified regex pattern.

`--match-path` *glob*
Only run tests in source files matching the specified glob pattern.

`--no-match-path` *glob*
Only run tests in source files that do not match the specified glob pattern.

`--debug` *regex*
Run a test in the debugger.

The argument passed to this flag is the name of the test function you want to run, and it works the same as `--match-test`.

If more than one test matches your specified criteria, you must add additional filters until only one test is found (see `--match-contract` and `--match-path`).

The matching test will be opened in the debugger regardless of the outcome of the test.

If the matching test is a fuzz test, then it will open the debugger on the first failure case. If the fuzz test does not fail, it will open the debugger on the last fuzz case.

For more fine-grained control of which fuzz case is run, see `forge debug`.

`--gas-report`
Print a gas report.

`--allow-failure`
Exit with code 0 even if a test fails.

`--fail-fast`
Stop running tests after the first failure.

`--etherscan-api-key` *key*
Etherscan API key. If set, traces are decoded using Etherscan if `--fork-url` is also set.
Environment: `ETHERSCAN_API_KEY`

EVM Options

`-f` *url*
`--rpc-url` *url*
`--fork-url` *url*
Fetch state over a remote endpoint instead of starting from an empty state.

If you want to fetch state from a specific block number, see `--fork-block-number`.

`--fork-block-number` *block*
Fetch state from a specific block number over a remote endpoint. See `--fork-url`.

`--fork-retry-backoff <BACKOFF>`
Initial retry backoff on encountering errors.

`--no-storage-caching`
Explicitly disables the use of RPC caching.

All storage slots are read entirely from the endpoint. See `--fork-url`.

`-v`
`--verbosity`
Verbosity of the EVM.

Pass multiple times to increase the verbosity (e.g. `-v`, `-vv`, `-vvv`).

Verbosity levels:
- 2: Print logs for all tests
- 3: Print execution traces for failing tests
- 4: Print execution traces for all tests, and setup traces for failing tests
- 5: Print execution and setup traces for all tests

`--sender` *address*
The address which will be executing tests

`--initial-balance` *balance*
The initial balance of deployed contracts

`--ffi`
Enables the FFI cheatcode

Executor Options

`--base-fee <FEE>`
`--block-base-fee-per-gas <FEE>`
The base fee in a block (in wei).

`--block-coinbase` *address*
The coinbase of the block.

`--block-difficulty` *difficulty*
The block difficulty.

`--block-gas-limit` *gas_limit*
The block gas limit.

`--block-number` *block*
The block number.

`--block-timestamp` *timestamp*
The timestamp of the block (in seconds).

`--chain-id` *chain_id*
The chain ID.

`--gas-limit` *gas_limit*
The block gas limit.

`--gas-price` *gas_price*
The gas price (in wei).

`--tx-origin` *address*
The transaction origin.

Cache Options

`--force`
Clear the cache and artifacts folder and recompile.

Linker Options

`--libraries` *libraries*
Set pre-linked libraries.

The parameter must be in the format `<remapped path to lib>:<library name>:<address>`, e.g. `src/Contract.sol:Library:0x...`.

Can also be set in your configuration file as `libraries = ["<path>:<lib name>:<address>"]`.

Compiler Options

`--optimize`
Activate the Solidity optimizer.

`--optimizer-runs` *runs*
The number of optimizer runs.

`--via-ir`
Use the Yul intermediate representation compilation pipeline.

`--revert-strings`
How to treat revert and require reason strings.

`--use` *solc_version*
Specify the solc version, or a path to a local solc, to build with.

Valid values are in the format `x.y.z`, `solc:x.y.z` or `path/to/solc`.

`--offline`
Do not access the network. Missing solc versions will not be installed.

`--no-auto-detect`
Do not auto-detect solc.

`--ignored-error-codes` *error_codes*
Ignore solc warnings by error code. The parameter is a comma-separated list of error codes.

`--extra-output` *selector*
Extra output to include in the contract's artifact.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--extra-output-files` *selector*
Extra output to write to separate files.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--evm-version` *version*
The target EVM version.

Project Options

`--build-info`
Generate build info files.

`--build-info-path` *path*
Output path to directory that build info files will be written to.

`--root` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`-C` *path*
`--contracts` *path*
The contracts source directory.
Environment: `DAPP_SRC`

`--lib-paths` *path*
The path to the library folder.

`-R` *remappings*
`--remappings` *remappings*
The project's remappings.

The parameter is a comma-separated list of remappings in the format `<source>=<dest>`.

`--cache-path` *path*
The path to the compiler cache.

`--config-path` *file*
Path to the config file.

`--hh`
`--hardhat`
This is a convenience flag, and is the same as passing `--contracts contracts --lib-paths node-modules`.

`-o` *path*
`--out` *path*
The project's artifacts directory.

`--silent`
Suppress all output.

Display Options

`-j`
`--json`
Print the deployment information as JSON.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Create a snapshot:

```
forge snapshot
```

2. Generate a diff:

```
forge snapshot --diff
```

3. Check that the snapshots match:

```
forge snapshot --check
```

SEE ALSO

forge, forge test

forge coverage

NAME

forge-coverage - Displays which parts of your code are covered by tests.

SYNOPSIS

`forge coverage` [*options*]

DESCRIPTION

Displays which parts of your code are covered by tests.

Options

Report Options

`--report` allows you to specify the report type to use for coverage. This flag can be used multiple times.

It has three different options and is set to `summary` by default.

`summary`
: Outputs a chart showing what percentage of your code is covered by tests.

`lcov`
: Creates a lcov.info file containing your coverage data in the root of your project's directory.

`debug`
: Outputs lines describing the location of uncovered code.

Common Options

`-h`
`--help`
: Prints help information.

Optimization Option

`--ir-minimum` allows you to run the coverage with `via-ir` enabled for the "minimum amount of optimization" necessary.

EXAMPLES

1. View summarized coverage:

```
forge coverage
```

2. Create lcov file with coverage data:

```
forge coverage --report lcov
```

3. Output uncovered code locations:

```
forge coverage --report debug
```

SEE ALSO

forge, forge test

Deploy Commands

- forge create
- forge verify-contract
- forge verify-check
- forge flatten

forge create

NAME

forge-create - Deploy a smart contract.

SYNOPSIS

`forge create` [*options*] *contract*

DESCRIPTION

Deploy a smart contract.

The path to the contract is in the format `<path>:<contract>`, e.g. `src/Contract.sol:Contract`.

You can specify constructor arguments with `--constructor-args`. Alternatively, you can specify a file containing space-separated constructor arguments with `--constructor-args-path`.

Dynamic linking is not supported: you should predeploy your libraries and manually specify their addresses (see `--libraries`).

> ℹ️ **Note**
>
> The `--constructor-args` flag must be positioned last in the command, since it takes multiple values.

OPTIONS

Build Options

`--constructor-args` *args...*
: The constructor arguments.

`--constructor-args-path` *file*
The path to a file containing the constructor arguments.

`--verify`
Verify contract after creation. Runs `forge verify-contract` with the appropriate parameters.

`--verifier` *name*
The verification provider. Available options: `etherscan`, `sourcify` & `blockscout`. Default: `etherscan`. Note: make sure you add "/api?" to the end of the Blockscout homepage explorer URL.

`--verifier-url` *url*
The optional verifier url for submitting the verification request.
Environment: `VERIFIER_URL`

`--unlocked`
Send via `eth_sendTransaction` using the `--from` argument or `$ETH_FROM` as sender.

Transaction Options

`--gas-limit` *gas_limit*
Gas limit for the transaction.

`--gas-price` *price*
Gas price for the transaction, or max fee per gas for EIP1559 transactions.

`--priority-gas-price` *price*
Max priority fee per gas for EIP1559 transactions.

`--value` *value*
Ether to send in the transaction.

Either specified as an integer (wei), or as a string with a unit, for example:
- `1ether`
- `10gwei`
- `0.01ether`

`--nonce` *nonce*
Nonce for the transaction.

`--legacy`
Send a legacy transaction instead of an EIP1559 transaction.

This is automatically enabled for common networks without EIP1559.

WALLET OPTIONS - RAW:

`-i`
`--interactive <NUM>`
Open an interactive prompt to enter your private key. Takes a value for the number of keys to enter.
Defaults to `0`.

`--mnemonic-derivation-path <PATHS>`
The wallet derivation path. Works with both `--mnemonic-path` and hardware wallets.

`--mnemonic-indexes <INDEXES>`
Use the private key from the given mnemonic index. Used with --mnemonic-paths.
Defaults to `0`.

`--mnemonic-passphrase <PASSPHRASE>`
Use a BIP39 passphrases for the mnemonic.

`--mnemonic <PATHS>`
Use the mnemonic phrases or mnemonic files at the specified paths.

`--private-key <RAW_PRIVATE_KEY>`
Use the provided private key.

`--private-keys <RAW_PRIVATE_KEYS>`
Use the provided private keys.

Wallet Options - Keystore

`--keystore` *path*
Use the keystore in the given folder or file.
Environment: `ETH_KEYSTORE`

`--account` *account-name*
Use a keystore from the default keystores folder (~/.foundry/keystores) by its filename.
Environment: `ETH_KEYSTORE_ACCOUNT`

`--interactive`

`--password` *password*
The keystore password. Used with `--keystore`. Environment: `ETH_PASSWORD`

Wallet Options - Hardware Wallet

`-t`
`--trezor`

Use a Trezor hardware wallet.

`-l`
`--ledger`
Use a Ledger hardware wallet.

Wallet Options - Remote

`-f` *address*
`--from` *address*
Sign the transaction with the specified account on the RPC.
Environment: `ETH_FROM`

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

`--flashbots`
Use the Flashbots RPC URL (https://rpc.flashbots.net).

Etherscan Options

`--chain` *chain_name*
The Etherscan chain.

`--etherscan-api-key` *key*
Etherscan API key, or the key of an Etherscan configuration table.
Environment: `ETHERSCAN_API_KEY`

Cache Options

`--force`
Clear the cache and artifacts folder and recompile.

Linker Options

`--libraries` *libraries*
Set pre-linked libraries.

The parameter must be in the format `<remapped path to lib>:<library name>:<address>`, e.g. `src/Contract.sol:Library:0x...`.

Can also be set in your configuration file as `libraries = ["<path>:<lib name>:<address>"]`.

Compiler Options

`--optimize`
Activate the Solidity optimizer.

`--optimizer-runs` *runs*
The number of optimizer runs.

`--via-ir`
Use the Yul intermediate representation compilation pipeline.

`--revert-strings`
How to treat revert and require reason strings.

`--use` *solc_version*
Specify the solc version, or a path to a local solc, to build with.

Valid values are in the format `x.y.z`, `solc:x.y.z` or `path/to/solc`.

`--offline`
Do not access the network. Missing solc versions will not be installed.

`--no-auto-detect`
Do not auto-detect solc.

`--ignored-error-codes` *error_codes*
Ignore solc warnings by error code. The parameter is a comma-separated list of error codes.

`--extra-output` *selector*
Extra output to include in the contract's artifact.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--extra-output-files` *selector*
Extra output to write to separate files.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--evm-version` *version*
The target EVM version.

Project Options

`--build-info`
Generate build info files.

`--build-info-path` *path*
Output path to directory that build info files will be written to.

`--root` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`-C` *path*
`--contracts` *path*
The contracts source directory.
Environment: `DAPP_SRC`

`--lib-paths` *path*
The path to the library folder.

`-R` *remappings*
`--remappings` *remappings*
The project's remappings.

The parameter is a comma-separated list of remappings in the format `<source>=<dest>`.

`--cache-path` *path*
The path to the compiler cache.

`--config-path` *file*
Path to the config file.

`--hh`
`--hardhat`
This is a convenience flag, and is the same as passing `--contracts contracts --lib-paths node-modules`.

`-o` *path*
`--out` *path*
The project's artifacts directory.

`--silent`
Suppress all output.

Display Options

`-j`
`--json`
Print the deployment information as JSON.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Deploy a contract with no constructor arguments:

```
forge create src/Contract.sol:ContractWithNoConstructor
```

2. Deploy a contract with two constructor arguments:

```
forge create src/Contract.sol:MyToken --constructor-args "My Token" "MT"
```

SEE ALSO

forge, forge build, forge verify-contract

forge verify-contract

NAME

forge-verify-contract - Verify smart contracts on a chosen verification provider.

SYNOPSIS

`forge verify-contract` [*options*] *address contract*

DESCRIPTION

Verifies a smart contract on a chosen verification provider.

You must provide:

- The contract address
- The contract name or the path to the contract (read below) In case of Etherscan verification, you must also provide:
- Your Etherscan API key, either by passing it as an argument or setting `ETHERSCAN_API_KEY`

To find the exact compiler version, run `~/.svm/x.y.z/solc-x.y.z --version` and search for the 8 hex digits in the version string here.

The path to the contract is in the format `<path>:<contract>`, e.g. `src/Contract.sol:Contract`.

By default, smart contracts are verified in a multi-file fashion. If you want to flatten the contract before verifying, pass `--flatten`.

This command will try to compile the source code of the flattened contract if `--flatten` is passed before verifying. If you do not want that, pass `--force`.

You can specify **ABI-encoded** constructor arguments with `--constructor-args`. Alternatively, you can specify a file containing **space-separated** constructor arguments with `--constructor-args-path`. (Note that cache must be enabled in the config for the latter to work.)

OPTIONS

Verify Contract Options

`--verifier` *name*
The verification provider. Available options: `etherscan`, `sourcify` & `blockscout`. Default: `etherscan`. Note: make sure you add "/api?" to the end of the Blockscout homepage explorer URL.

`--verifier-url` *url*
The optional verifier url for submitting the verification request.
Environment: `VERIFIER_URL`

`--compiler-version` *version*
The compiler version used to build the smart contract.

To find the exact compiler version, run `~/.svm/x.y.z/solc-x.y.z --version` where `x` and `y` are major and minor version numbers respectively, then search for the 8 hex digits in the version string here.

`--num-of-optimizations` *num*
`--optimizer-runs` *num*
The number of optimization runs used to build the smart contract.

`--constructor-args` *args*
The ABI-encoded constructor arguments. Conflicts with `--constructor-args-path`.

`--constructor-args-path` *file*
The path to a file containing the constructor arguments. Conflicts with `--constructor-args`.

`--chain-id` *chain*
`--chain` *chain*
The ID or name of the chain the contract is deployed to.
Default: mainnet

`--flatten`
Flag indicating whether to flatten the source code before verifying.

If this flag is not provided, the JSON standard input will be used instead.

`-f`
`--force`
Do not compile the flattened smart contract before verifying.

`--delay` *delay*
Optional timeout to apply in between attempts in seconds. Defaults to 3.

`--retries` *retries*
Number of attempts for retrying. Defaults to 15.

`--show-standard-json-input`
Command outputs JSON suitable for saving to a file and uploading to block explorers for verification.

`--watch`
Wait for verification result after submission.
Automatically runs `forge verify-check` until the verification either fails or succeeds.

Project Options

`--build-info`
Generate build info files.

`--build-info-path` *path*
Output path to directory that build info files will be written to.

`--root` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`-C` *path*
`--contracts` *path*
The contracts source directory.
Environment: `DAPP_SRC`

`--lib-paths` *path*
The path to the library folder.

`-R` *remappings*
`--remappings` *remappings*
The project's remappings.

The parameter is a comma-separated list of remappings in the format `<source>=<dest>`.

`--cache-path` *path*
The path to the compiler cache.

`--config-path` *file*
Path to the config file.

`--hh`
`--hardhat`
This is a convenience flag, and is the same as passing `--contracts contracts --lib-paths node-modules`.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Verify a contract with JSON standard input on Etherscan

```
forge verify-contract <address> SomeContract --watch
```

2. Verify a contract on a custom Sourcify instance

```
forge verify-contract --verifier sourcify \
  --verifier-url http://localhost:5000 <address> SomeContract
```

3. Verify a flattened contract built with solc v0.8.11+commit.d7f03943:

```
forge verify-contract --flatten --watch --compiler-version
"v0.8.11+commit.d7f03943" \
  --constructor-args $(cast abi-encode
"constructor(string,string,uint256,uint256)" "ForgeUSD" "FUSD" 18
1000000000000000000000) \
  <address> MyToken
```

4. Verify a flattened contract by specifying constructor arguments in a file:

```
forge verify-contract --flatten --watch --compiler-version
"v0.8.11+commit.d7f03943" \
  --constructor-args-path constructor-args.txt <address>
src/Token.sol:MyToken
```

where `constructor-args.txt` contains the following content:

```
ForgeUSD FUSD 18 1000000000000000000000
```

5. Verify a contract with Blockscout right after deployment (make sure you add "/api?" to the end of the Blockscout homepage explorer URL):

```
forge create --rpc-url <rpc_https_endpoint> --private-key
$devTestnetPrivateKey src/Contract.sol:SimpleStorage --verify --verifier
blockscout --verifier-url <blockscout_homepage_explorer_url>/api?
```

SEE ALSO

forge, forge create, forge flatten, forge verify-check

forge verify-check

NAME

forge-verify-check - Check verification status on a chosen verification provider.

SYNOPSIS

`forge verify-check` [*options*] *id* [*etherscan_key*]

The *id* is the verification identifier. For Etherscan & Bloxroute - it is the submission GUID, for Sourcify - it's the contract address.

DESCRIPTION

Check verification status on a chosen verification provider.

For Etherscan, you must provide an Etherscan API key, either by passing it as an argument or setting `ETHERSCAN_API_KEY`

OPTIONS

Verify Contract Options

`--verifier` *name*
The verification provider. Available options: `etherscan`, `sourcify` & `blockscout`. Default: `etherscan`. Note: make sure you add "/api?" to the end of the Blockscout homepage explorer URL.

`--verifier-url` *url*
The optional verifier url for submitting the verification request.
Environment: `VERIFIER_URL`

`--chain-id` *chain_id*
The chain ID the contract is deployed to (either a number or a chain name).
Default: mainnet

`--delay` *delay*
Optional timeout to apply in between attempts in seconds. Defaults to 3.

`--retries` *retries*

Number of attempts for retrying. Defaults to 15.

Common Options

`-h`

`--help`

Prints help information.

SEE ALSO

forge, forge create, forge verify-contract

forge flatten

NAME

forge-flatten - Flatten a source file and all of its imports into one file.

SYNOPSIS

`forge flatten` [*options*] *file*

DESCRIPTION

Flatten a source file and all of its imports into one file.

If `--output <FILE>` is not set, then the flattened contract will be output to stdout.

OPTIONS

Flatten Options

`-o` *file*
`--output` *file*
The path to output the flattened contract. If not specified, the flattened contract will be output to stdout.

Project Options

`--build-info`
Generate build info files.

`--build-info-path` *path*
Output path to directory that build info files will be written to.

`--root` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`-C` *path*
`--contracts` *path*
The contracts source directory.
Environment: `DAPP_SRC`

`--lib-paths` *path*
The path to the library folder.

`-R` *remappings*
`--remappings` *remappings*
The project's remappings.

The parameter is a comma-separated list of remappings in the format `<source>=<dest>`.

`--cache-path` *path*
The path to the compiler cache.

`--config-path` *file*
Path to the config file.

`--hh`
`--hardhat`
This is a convenience flag, and is the same as passing `--contracts contracts --lib-paths node-modules`.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Flatten `src/Contract.sol`:

```
forge flatten src/Contract.sol
```

2. Flatten `src/Contract.sol` and write the result to `src/Contract.flattened.sol`:

```
forge flatten --output src/Contract.flattened.sol src/Contract.sol
```

SEE ALSO

forge, forge verify-contract

Utility Commands

- forge debug
- forge bind
- forge cache
- forge cache clean
- forge cache ls
- forge script
- forge upload-selectors
- forge doc

forge debug

NAME

forge-debug - Debug a single smart contract as a script.

SYNOPSIS

`forge debug` [*options*] *path* [*args...*]

DESCRIPTION

Debugs a single smart contract located in the source file (*path*) as a script.

If multiple contracts are in the specified source file, you must pass `--target-contract` to specify what contract you want to run.

Calls

After the script is deployed to the internal EVM a call is made to a function with the signature `setUp()`, if present.

By default, the script is assumed to be contained in a function with the signature `run()`. If you wish to run a different function, pass `--sig <SIGNATURE>`.

The signature can be a fragment (`<function name>(<types>)`), or raw calldata.

If you pass a fragment, and the function has parameters, you can add the call parameters to the end of the command (*args*).

Forking

It is possible to run the script in a forked environment by passing `--fork-url <URL>`.

When the script is running in a forked environment, you can access all the state of the forked chain as you would if you had deployed the script. Cheatcodes are still available.

You can also specify a block number to fork from by passing `--fork-block-number <BLOCK>`. When forking from a specific block, the chain data is cached to `~/.foundry/cache`. If you do not want to cache the chain data, pass `--no-storage-caching`.

Debugging

It is possible to run the script in an interactive debugger. To start the debugger, pass `--debug`.

More information on the debugger can be found in the debugger chapter.

OPTIONS

Debug Options

`--target-contract` *contract_name*
: The name of the contract you want to run

`-s` *signature*
`--sig` *signature*
: The signature of the function you want to call in the contract, or raw calldata. Default: `run()`

`--debug`
: Open the script in the debugger.

EVM Options

`-f` *url*
`--rpc-url* *url*
`--fork-url` *url*
: Fetch state over a remote endpoint instead of starting from an empty state.

  If you want to fetch state from a specific block number, see `--fork-block-number`.

`--fork-block-number` *block*
: Fetch state from a specific block number over a remote endpoint. See `--fork-url`.

`--fork-retry-backoff <BACKOFF>`
: Initial retry backoff on encountering errors.

`--no-storage-caching`
: Explicitly disables the use of RPC caching.

  All storage slots are read entirely from the endpoint. See `--fork-url`.

`-v`
`--verbosity`
: Verbosity of the EVM.

Pass multiple times to increase the verbosity (e.g. `-v`, `-vv`, `-vvv`).

Verbosity levels:
- 2: Print logs for all tests
- 3: Print execution traces for failing tests
- 4: Print execution traces for all tests, and setup traces for failing tests
- 5: Print execution and setup traces for all tests

`--sender` *address*
The address which will be executing tests

`--initial-balance` *balance*
The initial balance of deployed contracts

`--ffi`
Enables the FFI cheatcode

Executor Options

`--base-fee <FEE>`
`--block-base-fee-per-gas <FEE>`
The base fee in a block (in wei).

`--block-coinbase` *address*
The coinbase of the block.

`--block-difficulty` *difficulty*
The block difficulty.

`--block-gas-limit` *gas_limit*
The block gas limit.

`--block-number` *block*
The block number.

`--block-timestamp` *timestamp*
The timestamp of the block (in seconds).

`--chain-id` *chain_id*
The chain ID.

`--gas-limit` *gas_limit*
The block gas limit.

`--gas-price` *gas_price*
The gas price (in wei).

`--tx-origin` *address*
The transaction origin.

Cache Options

`--force`
Clear the cache and artifacts folder and recompile.

Linker Options

`--libraries` *libraries*
Set pre-linked libraries.

The parameter must be in the format `<remapped path to lib>:<library name>:<address>`, e.g. `src/Contract.sol:Library:0x...`.

Can also be set in your configuration file as `libraries = ["<path>:<lib name>:<address>"]`.

Compiler Options

`--optimize`
Activate the Solidity optimizer.

`--optimizer-runs` *runs*
The number of optimizer runs.

`--via-ir`
Use the Yul intermediate representation compilation pipeline.

`--revert-strings`
How to treat revert and require reason strings.

`--use` *solc_version*
Specify the solc version, or a path to a local solc, to build with.

Valid values are in the format `x.y.z`, `solc:x.y.z` or `path/to/solc`.

`--offline`
Do not access the network. Missing solc versions will not be installed.

`--no-auto-detect`
Do not auto-detect solc.

`--ignored-error-codes` *error_codes*
Ignore solc warnings by error code. The parameter is a comma-separated list of error

codes.

`--extra-output` *selector*
Extra output to include in the contract's artifact.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--extra-output-files` *selector*
Extra output to write to separate files.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--evm-version` *version*
The target EVM version.

Project Options

`--build-info`
Generate build info files.

`--build-info-path` *path*
Output path to directory that build info files will be written to.

`--root` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`-C` *path*
`--contracts` *path*
The contracts source directory.
Environment: `DAPP_SRC`

`--lib-paths` *path*
The path to the library folder.

`-R` *remappings*
`--remappings` *remappings*
The project's remappings.

The parameter is a comma-separated list of remappings in the format `<source>=<dest>`.

`--cache-path` *path*
The path to the compiler cache.

`--config-path` *file*
Path to the config file.

`--hh`
`--hardhat`
This is a convenience flag, and is the same as passing `--contracts contracts --lib-paths node-modules`.

`-o` *path*
`--out` *path*
The project's artifacts directory.

`--silent`
Suppress all output.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Execute the `run()` function in a contract:

```
forge debug src/Contract.sol
```

2. Open a script in the debugger:

```
forge debug src/Contract.sol --debug
```

3. Execute the `foo()` function in a contract:

```
forge debug src/Contract.sol --sig "foo()"
```

4. Execute a contract with a function that takes parameters:

```
forge debug src/Contract.sol --sig "foo(string,uint256)" "hello" 100
```

5. Execute a contract with raw calldata:

```
forge debug src/Contract.sol --sig "0x..."
```

SEE ALSO

forge, forge test

forge bind

NAME

forge-bind - Generate Rust bindings for smart contracts.

SYNOPSIS

`forge bind` [*options*]

DESCRIPTION

Generates Rust bindings for smart contracts using ethers-rs.

The bindings are generated from the project's artifacts, which by default is `./out/`. If you want to generate bindings for artifacts in a different directory, pass `--bindings-path <PATH>`.

There are three output options:

- Generate bindings in a crate (default)
- Generate bindings in a module by passing `--module`
- Generate bindings in a single file by passing `--single-file`

By default, the command will check that existing bindings are correct and exit accordingly. You can overwrite the existing bindings by passing `--overwrite`.

OPTIONS

Project Options

`-b` *path*
`--bindings-path` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`--crate-name` *name*
The name of the Rust crate to generate, if you are generating a crate (default).
This should be a valid crates.io crate name.

Default: foundry-contracts

`--crate-version` *semver*

The version of the Rust crate to generate, if you are generating a crate (default).

This should be a standard semver version string.

Default: 0.0.1

`--module`

Generate the bindings as a module instead of a crate.

`--single-file`

Generate bindings as a single file.

`--overwrite`

Overwrite existing generated bindings. By default, the command will check that the bindings are correct, and then exit.

If `--overwrite` is passed, it will instead delete and overwrite the bindings.

`--root` *path*

The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`--skip-cargo-toml`

Skip Cargo.toml consistency checks.

This allows you to manage the ethers version without giving up on consistency checks.

An example would be if you use additional features of ethers like `ws`, `ipc`, or `rustls` and get an `ethers-providers` version mismatch.

`--skip-build`

Skips running forge build before generating binding.

This allows you to skip the default `forge build` step that's executed first and instead generate bindings using the already existing artifacts.

`--select-all`

By default all contracts ending with `Test` or `Script` are excluded. This will explicitly generate bindings for all contracts. Conflicts with `--select` and `--skip`.

`--select` *regex+*

Create bindings only for contracts whose names match the specified filter(s). Conflicts with `--skip`.

`--skip` *regex+*

Create bindings only for contracts whose names do not match the specified filter(s). Conflicts with `--select`.

Common Options

`-h`
`--help`
Prints help information.

Cache Options

`--force`
Clear the cache and artifacts folder and recompile.

Linker Options

`--libraries` *libraries*
Set pre-linked libraries.

The parameter must be in the format `<remapped path to lib>:<library name>:<address>`, e.g. `src/Contract.sol:Library:0x...`.

Can also be set in your configuration file as `libraries = ["<path>:<lib name>:<address>"]`.

Compiler Options

`--optimize`
Activate the Solidity optimizer.

`--optimizer-runs` *runs*
The number of optimizer runs.

`--via-ir`
Use the Yul intermediate representation compilation pipeline.

`--revert-strings`
How to treat revert and require reason strings.

`--use` *solc_version*
Specify the solc version, or a path to a local solc, to build with.

Valid values are in the format `x.y.z`, `solc:x.y.z` or `path/to/solc`.

`--offline`
Do not access the network. Missing solc versions will not be installed.

`--no-auto-detect`
Do not auto-detect solc.

`--ignored-error-codes` *error_codes*
Ignore solc warnings by error code. The parameter is a comma-separated list of error codes.

`--extra-output` *selector*
Extra output to include in the contract's artifact.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--extra-output-files` *selector*
Extra output to write to separate files.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--evm-version` *version*
The target EVM version.

Project Options

`--build-info`
Generate build info files.

`--build-info-path` *path*
Output path to directory that build info files will be written to.

`--root` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`-C` *path*
`--contracts` *path*
The contracts source directory.
Environment: `DAPP_SRC`

`--lib-paths` *path*
The path to the library folder.

`-R` *remappings*
`--remappings` *remappings*
The project's remappings.

The parameter is a comma-separated list of remappings in the format `<source>=<dest>`.

`--cache-path` *path*
: The path to the compiler cache.

`--config-path` *file*
: Path to the config file.

`--hh`
`--hardhat`
: This is a convenience flag, and is the same as passing `--contracts contracts --lib-paths node-modules`.

`-o` *path*
`--out` *path*
: The project's artifacts directory.

`--silent`
: Suppress all output.

SEE ALSO

forge

`forge cache`

NAME

forge-cache - Manage the Foundry cache.

SYNOPSIS

`forge cache` [*options*] *command* [*args*]
`forge cache` [*options*] `--version`
`forge cache` [*options*] `--help`

DESCRIPTION

This program is a set of tools to manage the Foundry cache.

COMMANDS

forge cache clean
: Cleans cached data from `~/.foundry`.

forge cache ls
: Shows cached data from `~/.foundry`.

OPTIONS

Special Options

`-V`
`--version`
: Print version info and exit.

Common Options

`-h`
`--help`

Prints help information.

`forge cache clean`

NAME

forge-cache-clean - Cleans cached data from `~/.foundry`.

SYNOPSIS

`forge cache clean` [*options*] [–] [*chains..*]

DESCRIPTION

Removes files in the `~/.foundry/cache` folder which is used to cache Etherscan verification status and block data.

OPTIONS

`-b`
`--blocks`
One or more block numbers separated by comma with no spaces

`--etherscan` A boolean flag that specifies to only remove the block explorer portion of the cache

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Remove the entire cache (also, `forge cache clean` is an alias for this)

```
forge cache clean all
```

2. Remove the entire block explorer cache

```
forge cache clean all --etherscan
```

3. Remove cache data for a specific chain, by name

```
forge cache clean rinkeby
```

4. Remove cache data for a specific block number on a specific chain. Does not work if `chain` is `all`

```
forge cache clean rinkeby -b 150000
```

5. Remove block explorer cache data for a specific chain. Does not work if `--blocks` are specified.

```
forge cache clean rinkeby --etherscan
```

6. Specify multiple chains

```
forge cache clean rinkeby mainnet
```

7. Specify multiple blocks

```
forge cache clean rinkeby --blocks 530000,9000000,9200000
```

SEE ALSO

forge, forge cache

`forge cache ls`

NAME

forge-cache-ls - Shows cached data from `~/.foundry`.

SYNOPSIS

`forge cache ls` [*chains..*]

DESCRIPTION

Lists what is in the `~/.foundry/cache` folder currently.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Show the entire cache (also, `forge cache ls` is an alias for this)

   ```
   forge cache ls all
   ```

2. Show cache data for a specific chain, by name

   ```
   forge cache ls rinkeby
   ```

3. Specify multiple chains

   ```
   forge cache ls rinkeby mainnet
   ```

SEE ALSO

forge, forge cache

forge script

NAME

forge-script - Run a smart contract as a script, building transactions that can be sent onchain.

SYNOPSIS

`forge script` [*options*] *path* [*args...*]

DESCRIPTION

Run a smart contract as a script, building transactions that can be sent onchain.

Scripts can be used to apply state transitions on live contracts, or deploy and initialize a complex set of smart contracts using Solidity.

OPTIONS

`--broadcast`
Broadcasts the transactions.

`--debug`
Open the script in the debugger. Takes precedence over broadcast.

`-g`
`--gas-estimate-multiplier` *multiplier*
Relative percentage by which to multiply all gas estimates. (i.e. set to 200 to double them)
Default: 130

`--json`
Output results in JSON format.
Note: The output is under development and prone to change.

`--legacy`
Use legacy transactions instead of EIP1559 ones. This is auto-enabled for common networks without EIP1559.

`--resume`
Resumes submitting transactions that failed or timed-out previously.

`-s`
`--sig` *signature*
The signature of the function you want to call in the contract, or raw calldata.
Default: `run()`

`--skip-simulation`
Skips on-chain simulation.

`--skip`
Skip compilation of non-essential contract directories like test or script (usage `--skip test`).

`--non-interactive`
Remove interactive prompts which appear if the contract is near the EIP-170 size limit.

`--slow`
Makes sure a transaction is sent, only after its previous one has been confirmed and succeeded.

`--target-contract` *contract_name*
The name of the contract you want to run.

`--priority-gas-price`
Sets the priority gas price for EIP1559 transactions. Useful for when gas prices are volatile and you want to get your transaction included.

`--with-gas-price` *price*
Sets the gas price for **broadcasted** legacy transactions, or the max fee for broadcasted EIP1559 transactions.
Note: To set the gas price in the execution environment of the script use `--gas-price` instead (see below).

Etherscan Options

`--chain` *chain_name*
The Etherscan chain.

`--etherscan-api-key` *key*
Etherscan API key, or the key of an Etherscan configuration table.
Environment: `ETHERSCAN_API_KEY`

Verification Options

`--verify`
If it finds a matching broadcast log, it tries to verify every contract found in the receipts.

`--verifier` *name*
The verification provider. Available options: `etherscan`, `sourcify` & `blockscout`. Default: `etherscan`. Note: make sure you add "/api?" to the end of the Blockscout homepage explorer URL.

`--verifier-url` *url*
The optional verifier url for submitting the verification request.
Environment: `VERIFIER_URL`

`--delay` *delay*
Optional timeout to apply in between attempts in seconds. Defaults to 3.

`--retries` *retries*
Number of attempts for retrying. Defaults to 15.

Cache Options

`--force`
Clear the cache and artifacts folder and recompile.

Linker Options

`--libraries` *libraries*
Set pre-linked libraries.

The parameter must be in the format `<remapped path to lib>:<library name>:<address>`, e.g. `src/Contract.sol:Library:0x...`.

Can also be set in your configuration file as `libraries = ["<path>:<lib name>:<address>"]`.

Compiler Options

`--optimize`
Activate the Solidity optimizer.

`--optimizer-runs` *runs*
The number of optimizer runs.

`--via-ir`
Use the Yul intermediate representation compilation pipeline.

`--revert-strings`
How to treat revert and require reason strings.

`--use` *solc_version*
Specify the solc version, or a path to a local solc, to build with.

Valid values are in the format `x.y.z`, `solc:x.y.z` or `path/to/solc`.

`--offline`
Do not access the network. Missing solc versions will not be installed.

`--no-auto-detect`
Do not auto-detect solc.

`--ignored-error-codes` *error_codes*
Ignore solc warnings by error code. The parameter is a comma-separated list of error codes.

`--extra-output` *selector*
Extra output to include in the contract's artifact.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--extra-output-files` *selector*
Extra output to write to separate files.

Example keys: `abi`, `storageLayout`, `evm.assembly`, `ewasm`, `ir`, `ir-optimized`, `metadata`.

For a full description, see the Solidity docs.

`--evm-version` *version*
The target EVM version.

Project Options

`--build-info`
Generate build info files.

`--build-info-path` *path*
Output path to directory that build info files will be written to.

`--root` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`-C` *path*
`--contracts` *path*
The contracts source directory.
Environment: `DAPP_SRC`

`--lib-paths` *path*
The path to the library folder.

`-R` *remappings*
`--remappings` *remappings*
The project's remappings.

The parameter is a comma-separated list of remappings in the format `<source>=<dest>`.

`--cache-path` *path*
The path to the compiler cache.

`--config-path` *file*
Path to the config file.

`--hh`
`--hardhat`
This is a convenience flag, and is the same as passing `--contracts contracts --lib-paths node-modules`.

`-o` *path*
`--out` *path*
The project's artifacts directory.

`--silent`
Suppress all output.

Build Options

`--names`
Print compiled contract names.

`--sizes`
Print compiled non-test contract sizes, exiting with code 1 if any of them are above the size limit.

Watch Options

`-w` [*path...*]
`--watch` [*path...*]
Watch specific file(s) or folder(s).

By default, the project's source directory is watched.

`-d` *delay*
`--delay` *delay*
File update debounce delay.

During the delay, incoming change events are accumulated and only once the delay has passed, is an action taken.
Note that this does not mean a command will be started: if `--no-restart` is given and a command is already running, the outcome of the action will be to do nothing.

Defaults to 50ms. Parses as decimal seconds by default, but using an integer with the `ms` suffix may be more convenient.

When using `--poll` mode, you'll want a larger duration, or risk overloading disk I/O.

`--no-restart`
Do not restart the command while it's running.

`--run-all`
Explicitly re-run the command on all files when a change is made.

Wallet Options - Raw

`-i --interactives` *num*
Open an interactive prompt to enter your private key. Takes a value for the number of keys to enter.
Default: 0

`--mnemonic-indexes` *indexes*
Use the private key from the given mnemonic index. Used with --mnemonic-path.
Default: 0

`--mnemonic-paths` *paths*
Use the mnemonic file at the specified path(s).

`--private-key` *raw_private_key*
Use the provided private key.

`--private-keys` *raw_private_keys*
Use the provided private keys.

Wallet Options - Keystore

`--keystores` *paths*
Use the keystores in the given folders or files.
Environment: `ETH_KEYSTORE`

`--account` *account-name*
Use a keystore from the default keystores folder (~/.foundry/keystores) by its filename.
Environment: `ETH_KEYSTORE_ACCOUNT`

`--password` *passwords*
The keystore passwords. Used with `--keystore`.

Wallet Options - Hardware Wallet

`-t`
`--trezor`
Use a Trezor hardware wallet.

`-l`
`--ledger`
Use a Ledger hardware wallet.

`--hd-paths` *paths*
The derivation paths to use with hardware wallets.

Wallet Options - Remote

`-a` *addresses* `--froms` *addresses* Sign the transaction with the specified accounts on the RPC. Environment: `ETH_FROM`

EVM Options

`-f` *url*
`--rpc-url` *url*
`--fork-url` *url*
Fetch state over a remote endpoint instead of starting from an empty state.

If you want to fetch state from a specific block number, see `--fork-block-number`.

`--fork-block-number` *block*
Fetch state from a specific block number over a remote endpoint. See `--fork-url`.

`--fork-retry-backoff <BACKOFF>`
Initial retry backoff on encountering errors.

`--no-storage-caching`
Explicitly disables the use of RPC caching.

All storage slots are read entirely from the endpoint. See `--fork-url`.

`-v`
`--verbosity`
Verbosity of the EVM.

Pass multiple times to increase the verbosity (e.g. `-v`, `-vv`, `-vvv`).

Verbosity levels:
- 2: Print logs for all tests
- 3: Print execution traces for failing tests
- 4: Print execution traces for all tests, and setup traces for failing tests
- 5: Print execution and setup traces for all tests

`--sender` *address*
The address which will be executing tests

`--initial-balance` *balance*
The initial balance of deployed contracts

`--ffi`
Enables the FFI cheatcode

Executor Options

`--base-fee <FEE>`
`--block-base-fee-per-gas <FEE>`
The base fee in a block (in wei).

`--block-coinbase` *address*
The coinbase of the block.

`--block-difficulty` *difficulty*
The block difficulty.

`--block-gas-limit` *gas_limit*
The block gas limit.

`--block-number` *block*
The block number.

`--block-timestamp` *timestamp*
The timestamp of the block (in seconds).

`--chain-id` *chain_id*
The chain ID.

`--gas-limit` *gas_limit*
The block gas limit.

`--gas-price` *gas_price*
The gas price (in wei).

`--tx-origin` *address*
The transaction origin.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Run `BroadcastTest` as a script, broadcasting generated transactions on-chain

```
forge script ./test/Broadcast.t.sol --tc BroadcastTest --sig "deploy()" \
    -vvv --fork-url $SEPOLIA_RPC_URL
```

2. Deploy a contract on Polygon (see scripting tutorial for an example script). *The verifier url is different for every network.*

```
forge script script/NFT.s.sol:MyScript --chain-id 137 --rpc-url $RPC_URL \
    --etherscan-api-key $POLYGONSCAN_API_KEY --verifier-url
https://api.polygonscan.com/api \
    --broadcast --verify -vvvv
```

3. Resume a failed script. Using the above as an example, remove `--broadcast` add `--resume`

```
forge script script/NFT.s.sol:MyScript --chain-id 137 --rpc-url $RPC_URL \
    --etherscan-api-key $POLYGONSCAN_API_KEY --verifier-url
https://api.polygonscan.com/api \
    --verify -vvvv --resume
```

4. Verify contracts that were just deployed with a script

```
forge script script/NFT.s.sol --rpc-url $RPC_URL --verify --resume
```

forge upload-selectors

NAME

forge-upload-selectors - Uploads abi of given contract to https://sig.eth.samczsun.com function selector database.

SYNOPSIS

`forge upload-selectors` [*options*] *contract*

DESCRIPTION

Uploads abi of given contract to https://sig.eth.samczsun.com function selector database.

OPTIONS

Project Options

`--build-info`
Generate build info files.

`--build-info-path` *path*
Output path to directory that build info files will be written to.

`--root` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`-C` *path*
`--contracts` *path*
The contracts source directory.
Environment: `DAPP_SRC`

`--lib-paths` *path*
The path to the library folder.

`-R` *remappings*
`--remappings` *remappings*

The project's remappings.

The parameter is a comma-separated list of remappings in the format `<source>=<dest>`.

`--cache-path` *path*
The path to the compiler cache.

`--config-path` *file*
Path to the config file.

`--hh`
`--hardhat`
This is a convenience flag, and is the same as passing `--contracts contracts --lib-paths node-modules`.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Upload ABI to selector database

```
forge upload-selectors LinearVestingVault
```

forge doc

NAME

forge-doc - Generate documentation for Solidity source files.

SYNOPSIS

`forge doc` [*options*]

DESCRIPTION

Generates and builds an mdbook from Solidity source files.

OPTIONS

`--root` *path*
The project's root path. By default, this is the root directory of the current git repository, or the current working directory.

`--out` *path* The output path for the generated mdbook. By default, it is the `docs/` in project root.

`--build` Build the `mdbook` from generated files.

`--serve` Serve the documentation locally.

`--hostname` *hostname* Hostname for serving documentation. Requires `--serve`.

`--port` *port* Port for serving documentation. Requires `--serve`.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Generate documentation.

```
forge doc
```

2. Generate and build documentation with specified output directory.

```
forge doc --build --out ./documentation
```

3. Generate and serve documentation locally on port 4000.

```
forge doc --serve --port 4000
```

SEE ALSO

Doc config

cast Commands

- General Commands
- Chain Commands
- Transaction Commands
- Block Commands
- Account Commands
- ENS Commands
- Etherscan Commands
- ABI Commands
- Conversion Commands
- Utility Commands
- Wallet Commands

General Commands

- cast
- cast help
- cast completions

cast

NAME

cast - Perform Ethereum RPC calls from the comfort of your command line.

SYNOPSIS

`cast` [*options*] *command* [*args*] `cast` [*options*] `--version` `cast` [*options*] `--help`

DESCRIPTION

This program is a set of tools to interact with Ethereum and perform conversions.

COMMANDS

General Commands

cast help Display help information about Cast.

cast completions Generate shell autocompletions for Cast.

Chain Commands

cast chain-id Get the Ethereum chain ID.

cast chain Get the symbolic name of the current chain.

cast client Get the current client version.

Transaction Commands

cast publish Publish a raw transaction to the network.

cast receipt Get the transaction receipt for a transaction.

cast send Sign and publish a transaction.

cast call Perform a call on an account without publishing a transaction.

cast rpc Perform a raw JSON-RPC request [aliases: rp]

cast tx Get information about a transaction.

cast run Runs a published transaction in a local environment and prints the trace.

cast estimate Estimate the gas cost of a transaction.

cast access-list Create an access list for a transaction.

cast logs Get logs by signature or topic

Block Commands

cast find-block Get the block number closest to the provided timestamp.

cast gas-price Get the current gas price.

cast block-number Get the latest block number.

cast basefee Get the basefee of a block.

cast block Get information about a block.

cast age Get the timestamp of a block.

Account Commands

cast balance Get the balance of an account in wei.

cast storage Get the raw value of a contract's storage slot.

cast proof Generate a storage proof for a given storage slot.

cast nonce Get the nonce for an account.

cast code Get the bytecode of a contract.

cast codesize Get the runtime bytecode size of a contract.

ENS Commands

cast lookup-address Perform an ENS reverse lookup.

cast resolve-name Perform an ENS lookup.

cast namehash Calculate the ENS namehash of a name.

Etherscan Commands

cast etherscan-source Get the source code of a contract from Etherscan.

ABI Commands

cast abi-decode Decode ABI-encoded input or output data.

cast abi-encode ABI encode the given function arguments, excluding the selector.

cast 4byte Get the function signatures for the given selector from https://sig.eth.samczsun.com.

cast 4byte-decode Decode ABI-encoded calldata using https://sig.eth.samczsun.com.

cast 4byte-event Get the event signature for a given topic 0 from https://sig.eth.samczsun.com.

cast calldata ABI-encode a function with arguments.

cast calldata-decode Decode ABI-encoded input data.

cast pretty-calldata Pretty print calldata.

cast selectors Extracts function selectors and arguments from bytecode

cast upload-signature Upload the given signatures to https://sig.eth.samczsun.com.

Conversion Commands

cast format-bytes32-string Formats a string into bytes32 encoding.

cast from-bin Convert binary data into hex data.

cast from-fixed-point Convert a fixed point number into an integer.

cast from-utf8 Convert UTF8 to hex.

cast from-wei Convert wei into an ETH amount

cast parse-bytes32-address Parses a checksummed address from bytes32 encoding.

cast parse-bytes32-string Parses a string from bytes32 encoding.

cast to-ascii Convert hex data to an ASCII string.

cast to-base Convert a number of one base to another.

cast to-bytes32 Right-pads hex data to 32 bytes.

cast to-dec Converts a number of one base to decimal

cast to-fixed-point Convert an integer into a fixed point number.

cast to-hex Converts a number of one base to another

cast to-hexdata Normalize the input to lowercase, 0x-prefixed hex.

cast to-int256 Convert a number to a hex-encoded int256.

cast to-rlp RLP encodes hex data, or an array of hex data

cast to-uint256 Convert a number to a hex-encoded uint256.

cast to-unit Convert an ETH amount into another unit (ether, gwei, wei).

cast to-wei Convert an ETH amount to wei.

cast shl Perform a left shifting operation.

cast shr Perform a right shifting operation.

Utility Commands

cast sig Get the selector for a function.

cast sig-event Generate event signatures from event string.

cast keccak Hash arbitrary data using keccak-256.

cast compute-address Compute the contract address from a given nonce and deployer address.

cast create2 Generate a deterministic contract address using CREATE2

cast interface Generate a Solidity interface from a given ABI.

cast index Compute the storage slot for an entry in a mapping.

cast concat-hex Concatenate hex strings.

cast max-int Get the maximum i256 value.

cast min-int Get the minimum i256 value.

cast max-uint Get the maximum u256 value.

cast to-check-sum-address Convert an address to a checksummed format (EIP-55).

Wallet Commands

cast wallet Wallet management utilities.

cast wallet new Create a new random keypair.

cast wallet address Convert a private key to an address.

cast wallet sign Sign a message.

cast wallet vanity Generate a vanity address.

cast wallet verify Verify the signature of a message.

OPTIONS

Special Options

`-V --version` Print version info and exit.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Call a function on a contract:

```
cast call 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2 \
  "balanceOf(address)(uint256)" 0x...
```

2. Decode raw calldata:

```
cast calldata-decode "transfer(address,uint256)" \

0xa9059cbb000000000000000000000000e78388b4ce79068e89bf8aa7f218ef6b9ab0e9d0
0000000000000000000000000000000000000000000000008a8e4b1a3d8000
```

3. Encode calldata:

```
cast calldata "someFunc(address,uint256)" 0x... 1
```

BUGS

See https://github.com/foundry-rs/foundry/issues for issues.

cast help

NAME

cast-help - Get help for a Cast command

SYNOPSIS

`cast help` [*subcommand*]

DESCRIPTION

Prints a help message for the given command.

EXAMPLES

1. Get help for a command:

```
cast help call
```

2. Help is also available with the `--help` flag:

```
cast call --help
```

SEE ALSO

cast

cast completions

NAME

cast-completions - Generate shell completions script

SYNOPSIS

`cast completions` *shell*

DESCRIPTION

Generates a shell completions script for the given shell.

Supported shells are:

- bash
- elvish
- fish
- powershell
- zsh

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Generate shell completions script for zsh:

```
cast completions zsh > $HOME/.oh-my-zsh/completions/_cast
```

SEE ALSO

cast

Chain Commands

- cast chain-id
- cast chain
- cast client

cast chain-id

NAME

cast-chain-id - Get the Ethereum chain ID.

SYNOPSIS

`cast chain-id` [*options*]

DESCRIPTION

Get the Ethereum chain ID from the RPC endpoint we are connected to.

OPTIONS

RPC Options

`--rpc-url` *url*
: The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Get the chain ID when talking to `$RPC`:

```
cast chain-id --rpc-url $RPC
```

2. Get the chain ID when `$ETH_RPC_URL` is set:

```
cast chain-id
```

SEE ALSO

cast, cast chain

cast chain

NAME

cast-chain - Get the symbolic name of the current chain.

SYNOPSIS

`cast chain` [*options*]

DESCRIPTION

Get the symbolic chain name from the RPC endpoint we are connected to.

OPTIONS

RPC Options

`--rpc-url` *url*
: The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Get the chain name when talking to `$RPC`:

```
cast chain --rpc-url $RPC
```

2. Get the chain name when `$ETH_RPC_URL` is set:

```
cast chain
```

SEE ALSO

cast, cast chain-id

cast client

NAME

cast-client - Get the current client version.

SYNOPSIS

`cast client` [*options*]

DESCRIPTION

Get the current client version.

OPTIONS

RPC Options

`--rpc-url` *url*
: The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Get the current client version:

```
cast client
```

SEE ALSO

cast

Transaction Commands

- cast publish
- cast receipt
- cast send
- cast mktx
- cast call
- cast rpc
- cast tx
- cast run
- cast estimate
- cast access-list
- cast logs

cast publish

NAME

cast-publish - Publish a raw transaction to the network.

SYNOPSIS

`cast publish` [*options*] *tx*

DESCRIPTION

Publish a raw pre-signed transaction to the network.

OPTIONS

Publish Options

`--async`
`--cast-async`
Do not wait for a transaction receipt.
Environment: `CAST_ASYNC`

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

`--flashbots`
Use the Flashbots RPC URL (https://rpc.flashbots.net).

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Publish a pre-signed transaction:

```
cast publish --rpc-url $RPC $TX
```

2. Publish a pre-signed transaction with flashbots.

```
cast publish --flashbots $TX
```

SEE ALSO

cast, cast call, cast send, cast receipt, cast mktx

cast receipt

NAME

cast-receipt - Get the transaction receipt for a transaction.

SYNOPSIS

`cast receipt` [*options*] *tx_hash* [*field*]

DESCRIPTION

Get the transaction receipt for a transaction.

If *field* is specified, then only the given field of the receipt is displayed.

OPTIONS

Receipt Options

`--async`
`--cast-async`
Do not wait for the transaction receipt if it does not exist yet.
Environment: `CAST_ASYNC`

`-c` *confirmations*
`--confirmations` *confirmations*
Wait a number of confirmations before exiting. Default: `1`.

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Display Options

`-j`
`--json`
Print the deployment information as JSON.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Get a transaction receipt:

```
cast receipt $TX_HASH
```

2. Get the block number the transaction was included in:

```
cast receipt $TX_HASH blockNumber
```

SEE ALSO

cast, cast call, cast send, cast publish

cast send

NAME

cast-send - Sign and publish a transaction.

SYNOPSIS

`cast send` [*options*] *to* [*sig*] [*args...*]

DESCRIPTION

Sign and publish a transaction.

The destination (*to*) can be an ENS name or an address.

The signature (*sig*) can be:

- A fragment: `someFunction(uint256,bytes32)`
- A selector and encoded calldata: `0xcdba2fd4000000000000000000000000000000000000000000000000000000000000007a69`
- Only the function name: in this case Cast will try to fetch the function signature from Etherscan

OPTIONS

Transaction Options

`--gas-limit` *gas_limit*
Gas limit for the transaction.

`--gas-price` *price*
Gas price for the transaction, or max fee per gas for EIP1559 transactions.

`--priority-gas-price` *price*
Max priority fee per gas for EIP1559 transactions.

`--value` *value*
Ether to send in the transaction.

Either specified as an integer (wei), or as a string with a unit, for example:
- `1ether`
- `10gwei`
- `0.01ether`

`--nonce` *nonce*
Nonce for the transaction.

`--legacy`
Send a legacy transaction instead of an EIP1559 transaction.

This is automatically enabled for common networks without EIP1559.

`--resend`
Reuse the latest nonce of the sending account.

`--create` *code* [*sig args...*]
Deploy a contract by specifying raw bytecode, in place of specifying a *to* address.

Receipt Options

`--async`
`--cast-async`
Do not wait for the transaction receipt if it does not exist yet.
Environment: `CAST_ASYNC`

`-c` *confirmations*
`--confirmations` *confirmations*
Wait a number of confirmations before exiting. Default: `1`.

WALLET OPTIONS - RAW:

`-i`
`--interactive <NUM>`
Open an interactive prompt to enter your private key. Takes a value for the number of keys to enter.
Defaults to `0`.

`--mnemonic-derivation-path <PATHS>`
The wallet derivation path. Works with both `--mnemonic-path` and hardware wallets.

`--mnemonic-indexes <INDEXES>`
Use the private key from the given mnemonic index. Used with --mnemonic-paths.
Defaults to `0`.

`--mnemonic-passphrase <PASSPHRASE>`
Use a BIP39 passphrases for the mnemonic.

`--mnemonic <PATHS>`
Use the mnemonic phrases or mnemonic files at the specified paths.

`--private-key <RAW_PRIVATE_KEY>`
Use the provided private key.

`--private-keys <RAW_PRIVATE_KEYS>`
Use the provided private keys.

Wallet Options - Keystore

`--keystore` *path*
Use the keystore in the given folder or file.
Environment: `ETH_KEYSTORE`

`--account` *account-name*
Use a keystore from the default keystores folder (~/.foundry/keystores) by its filename.
Environment: `ETH_KEYSTORE_ACCOUNT`

`--interactive`

`--password` *password*
The keystore password. Used with `--keystore`. Environment: `ETH_PASSWORD`

Wallet Options - Hardware Wallet

`-t`
`--trezor`
Use a Trezor hardware wallet.

`-l`
`--ledger`
Use a Ledger hardware wallet.

Wallet Options - Remote

`-f` *address*
`--from` *address*
Sign the transaction with the specified account on the RPC.
Environment: `ETH_FROM`

`--unlocked`
Send via `eth_sendTransaction` using the `--from` argument or `$ETH_FROM` as sender.

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

`--flashbots`
Use the Flashbots RPC URL (https://rpc.flashbots.net).

Etherscan Options

`--chain` *chain_name*
The Etherscan chain.

`--etherscan-api-key` *key*
Etherscan API key, or the key of an Etherscan configuration table.
Environment: `ETHERSCAN_API_KEY`

Display Options

`-j`
`--json`
Print the deployment information as JSON.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Send some ether to Vitalik using your Ledger:

```
cast send --ledger vitalik.eth --value 0.1ether
```

2. Call `deposit(address token, uint256 amount)` on a contract:

```
cast send --ledger 0x... "deposit(address,uint256)" 0x... 1
```

3. Call a function that expects a `struct`:

```
contract Test {
    struct MyStruct {
        address addr;
        uint256 amount;
    }
    function myfunction(MyStruct memory t) public pure {}
}
```

Structs are encoded as tuples (see struct encoding)

```
cast send 0x... "myfunction((address,uint256))" "(0x...,1)"
```

4. Send a transaction with hex data in the `input` field of the transaction object:

```
cast send 0x... 0x68656c6c6f20776f726c64
```

SEE ALSO

cast, cast call, cast publish, cast receipt, cast mktx, struct encoding

cast mktx

NAME

cast-mktx - Build and sign a transaction.

SYNOPSIS

`cast mktx` [*options*] *to* [*sig*] [*args...*]

DESCRIPTION

Build and sign a transaction, without publishing it.

The destination (*to*) can be an ENS name or an address.

The signature (*sig*) can be:

- A fragment: `someFunction(uint256,bytes32)`
- A selector and encoded calldata: `0xcdba2fd4000000000000000000000000000000000000000000000000000000000000007a69`
- Only the function name: in this case Cast will try to fetch the function signature from Etherscan

OPTIONS

Transaction Options

`--gas-limit` *gas_limit*
Gas limit for the transaction.

`--gas-price` *price*
Gas price for the transaction, or max fee per gas for EIP1559 transactions.

`--priority-gas-price` *price*
Max priority fee per gas for EIP1559 transactions.

`--value` *value*
Ether to send in the transaction.

Either specified as an integer (wei), or as a string with a unit, for example:

- `1ether`
- `10gwei`
- `0.01ether`

`--nonce` *nonce*

Nonce for the transaction.

`--legacy`

Send a legacy transaction instead of an EIP1559 transaction.

This is automatically enabled for common networks without EIP1559.

`--create` *code* [*sig args...*]

Deploy a contract by specifying raw bytecode, in place of specifying a *to* address.

WALLET OPTIONS - RAW:

`-i`
`--interactive <NUM>`

Open an interactive prompt to enter your private key. Takes a value for the number of keys to enter.

Defaults to `0`.

`--mnemonic-derivation-path <PATHS>`

The wallet derivation path. Works with both `--mnemonic-path` and hardware wallets.

`--mnemonic-indexes <INDEXES>`

Use the private key from the given mnemonic index. Used with --mnemonic-paths.

Defaults to `0`.

`--mnemonic-passphrase <PASSPHRASE>`

Use a BIP39 passphrases for the mnemonic.

`--mnemonic <PATHS>`

Use the mnemonic phrases or mnemonic files at the specified paths.

`--private-key <RAW_PRIVATE_KEY>`

Use the provided private key.

`--private-keys <RAW_PRIVATE_KEYS>`

Use the provided private keys.

Wallet Options - Keystore

`--keystore` *path*

Use the keystore in the given folder or file.

Environment: `ETH_KEYSTORE`

`--account` *account-name*
Use a keystore from the default keystores folder (~/.foundry/keystores) by its filename.
Environment: `ETH_KEYSTORE_ACCOUNT`

`--interactive`

`--password` *password*
The keystore password. Used with `--keystore`. Environment: `ETH_PASSWORD`

Wallet Options - Hardware Wallet

`-t`
`--trezor`
Use a Trezor hardware wallet.

`-l`
`--ledger`
Use a Ledger hardware wallet.

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

`--flashbots`
Use the Flashbots RPC URL (https://rpc.flashbots.net).

Etherscan Options

`--chain` *chain_name*
The Etherscan chain.

`--etherscan-api-key` *key*
Etherscan API key, or the key of an Etherscan configuration table.
Environment: `ETHERSCAN_API_KEY`

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Sign a transaction that sends some ether to Vitalik using your Ledger:

```
cast mktx --ledger vitalik.eth --value 0.1ether
```

2. Sign a transaction that calls `deposit(address token, uint256 amount)` on a contract:

```
cast mktx --ledger 0x... "deposit(address,uint256)" 0x... 1
```

3. Sign a transaction that calls a function that expects a `struct`:

```
contract Test {
    struct MyStruct {
        address addr;
        uint256 amount;
    }
    function myfunction(MyStruct memory t) public pure {}
}
```

Structs are encoded as tuples (see struct encoding)

```
cast mktx 0x... "myfunction((address,uint256))" "(0x...,1)"
```

4. Sign a transaction with hex data in the `input` field of the transaction object:

```
cast mktx 0x... 0x68656c6c6f20776f726c64
```

SEE ALSO

cast, cast publish, cast send, struct encoding

cast call

NAME

cast-call - Perform a call on an account without publishing a transaction.

SYNOPSIS

`cast call` [*options*] *to sig* [*args...*]

DESCRIPTION

Perform a call on an account without publishing a transaction.

The destination (*to*) can be an ENS name or an address.

The signature (*sig*) can be:

- A fragment: `someFunction(uint256,bytes32)`
- A selector and encoded calldata: `0xcdba2fd4000000000000000000000000000000000000000000000000000000000000007a69`
- Only the function name: in this case Cast will try to fetch the function signature from Etherscan

OPTIONS

`--trace`
  Prints traces for the transaction.

`--debug`
  Opens an interactive debugger with the transaction. Needs `--trace`.

`--labels <address:label>`
  Labels to apply to the traces, with the format `address:label`. Needs `--trace`.

`--evm-version`
  The EVM version to use. Needs `--trace`.

Query Options

`-B` *block*
`--block` *block*
The block height you want to query at.

Can be a block number, or any of the tags: `earliest`, `finalized`, `safe`, `latest` or `pending`.

WALLET OPTIONS - RAW:

`-i`
`--interactive <NUM>`
Open an interactive prompt to enter your private key. Takes a value for the number of keys to enter.
Defaults to `0`.

`--mnemonic-derivation-path <PATHS>`
The wallet derivation path. Works with both `--mnemonic-path` and hardware wallets.

`--mnemonic-indexes <INDEXES>`
Use the private key from the given mnemonic index. Used with --mnemonic-paths.
Defaults to `0`.

`--mnemonic-passphrase <PASSPHRASE>`
Use a BIP39 passphrases for the mnemonic.

`--mnemonic <PATHS>`
Use the mnemonic phrases or mnemonic files at the specified paths.

`--private-key <RAW_PRIVATE_KEY>`
Use the provided private key.

`--private-keys <RAW_PRIVATE_KEYS>`
Use the provided private keys.

Wallet Options - Keystore

`--keystore` *path*
Use the keystore in the given folder or file.
Environment: `ETH_KEYSTORE`

`--account` *account-name*
Use a keystore from the default keystores folder (~/.foundry/keystores) by its filename.
Environment: `ETH_KEYSTORE_ACCOUNT`

`--interactive`

`--password` *password*
The keystore password. Used with `--keystore`. Environment: `ETH_PASSWORD`

Wallet Options - Hardware Wallet

`-t`
`--trezor`
Use a Trezor hardware wallet.

`-l`
`--ledger`
Use a Ledger hardware wallet.

Wallet Options - Remote

`-f` *address*
`--from` *address*
Sign the transaction with the specified account on the RPC.
Environment: `ETH_FROM`

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

`--flashbots`
Use the Flashbots RPC URL (https://rpc.flashbots.net).

Etherscan Options

`--chain` *chain_name*
The Etherscan chain.

`--etherscan-api-key` *key*
Etherscan API key, or the key of an Etherscan configuration table.
Environment: `ETHERSCAN_API_KEY`

Common Options

`-h`
`--help`

Prints help information.

EXAMPLES

1. Call `balanceOf(address)` on the WETH contract:

```
cast call 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2 \
  "balanceOf(address)(uint256)" 0x...
```

2. Call `tokenURI(uint256)(string)` on the Tubby Cats NFT contract:

```
export CONTRACT=0xca7ca7bcc765f77339be2d648ba53ce9c8a262bd
export TOKEN_ID=19938
cast call $CONTRACT "tokenURI(uint256)(string)" $TOKEN_ID
```

3. Call `getAmountsOut(uint,address[])` on the Uniswap v2 router contract:

```
cast call 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D \
 "getAmountsOut(uint,address[])" 1 "[0x6b...0f,0xc0...c2]"
```

SEE ALSO

cast, cast send, cast publish, cast receipt

cast rpc

NAME

cast-rpc - Perform a raw JSON-RPC request

SYNOPSIS

`cast rpc` [*options*] *METHOD* [*PARAMS...*]

DESCRIPTION

Perform a simple JSON-RPC POST request for the given method and with the params

OPTIONS

Query Options

`-r` *url*
`--rpc-url` *url*
The URL of the provider

`-w`
`--raw`
Pass the "params" as is If –raw is passed the first PARAM will be taken as the value of "params". If no params are given, stdin will be used. For example: rpc eth_getBlockByNumber '["0x123", false]' –raw => {"method": "eth_getBlockByNumber", "params": ["0x123", false] ... }

EXAMPLES

1. Get latest `eth_getBlockByNumber` on localhost:

```
cast rpc eth_getBlockByNumber "latest" "false"
```

2. Get `eth_getTransactionByHash` on localhost:

```
cast rpc eth_getTransactionByHash
0x2642e960d3150244e298d52b5b0f024782253e6d0b2c9a01dd4858f7b4665a3f
```

3. Get latest `eth_getBlockByNumber` on etherum mainnet:

```
cast rpc --rpc-url https://mainnet.infura.io/v3/ eth_getBlockByNumber
"latest" "false"
```

cast tx

NAME

cast-tx - Get information about a transaction.

SYNOPSIS

`cast tx` [*options*] *tx_hash* [*field*]

DESCRIPTION

Get information about a transaction.

If *field* is specified, then only the given field of the transaction is displayed.

OPTIONS

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Display Options

`-j`
`--json`
Print the deployment information as JSON.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Get information about a transaction:

```
cast tx $TX_HASH
```

2. Get the sender of a transaction:

```
cast tx $TX_HASH from
```

SEE ALSO

cast, cast receipt

cast run

NAME

cast-run - Runs a published transaction in a local environment and prints the trace.

SYNOPSIS

`cast run` [*options*] `--rpc-url` *url tx_hash*

DESCRIPTION

Runs a published transaction in a local environment and prints the trace.

By default, all transactions in the block prior to the transaction you want to replay are also replayed. If you want a quicker result, you can use `--quick`, however, results may differ from the live execution.

You can also open the transaction in a debugger by passing `--debug`.

OPTIONS

Run Options

`--label` *label*
: Labels an address in the trace.
The format is `<address>:<label>`. Can be passed multiple times.

`-q`
`--quick`
: Executes the transaction only with the state from the previous block.
May result in different results than the live execution!

`-v`
`--verbose`
: Addresses are fully displayed instead of being truncated.

`-d`
`--debug`

Open the script in the debugger.

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Replay a transaction (a simple transfer):

```
cast run
0xd15e0237413d7b824b784e1bbc3926e52f4726e5e5af30418803b8b327b4f8ca
```

2. Replay a transaction, applied on top of the state of the previous block:

```
cast run --quick \
  0xd15e0237413d7b824b784e1bbc3926e52f4726e5e5af30418803b8b327b4f8ca
```

3. Replay a transaction with address labels:

```
cast run \
  --label 0xc564ee9f21ed8a2d8e7e76c085740d5e4c5fafbe:sender \
  --label 0x40950267d12e979ad42974be5ac9a7e452f9505e:recipient \
  --label 0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2:weth \
  0xd15e0237413d7b824b784e1bbc3926e52f4726e5e5af30418803b8b327b4f8ca
```

4. Replay a transaction in the debugger:

```
cast run --debug \
  0xd15e0237413d7b824b784e1bbc3926e52f4726e5e5af30418803b8b327b4f8ca
```

SEE ALSO

cast

cast estimate

NAME

cast-estimate - Estimate the gas cost of a transaction.

SYNOPSIS

`cast estimate` [*options*] *to sig* [*args...*]

DESCRIPTION

Estimate the gas cost of a transaction.

The destination (*to*) can be an ENS name or an address.

The signature (*sig*) can be:

- A fragment: `someFunction(uint256,bytes32)`
- A selector and encoded calldata: `0xcdba2fd4000000000000000000000000000000000000000000000000000000000000007a69`
- Only the function name: in this case Cast will try to fetch the function signature from Etherscan

OPTIONS

Transaction Options

`--value` *value*

Ether to send in the transaction.

Either specified as an integer (wei), or as a string with a unit, for example:
- `1ether`
- `10gwei`
- `0.01ether`

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

`--flashbots`
Use the Flashbots RPC URL (https://rpc.flashbots.net).

Etherscan Options

`--chain` *chain_name*
The Etherscan chain.

`--etherscan-api-key` *key*
Etherscan API key, or the key of an Etherscan configuration table.
Environment: `ETHERSCAN_API_KEY`

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Estimate the gas cost of calling `deposit()` on the WETH contract:

```
cast estimate 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2 \
  --value 0.1ether "deposit()"
```

SEE ALSO

cast, cast send, cast publish, cast receipt

cast access-list

NAME

cast-access-list - Create an access list for a transaction.

SYNOPSIS

`cast access-list` [*options*] *to sig* [*args...*]

DESCRIPTION

Create an access list for a transaction.

The destination (*to*) can be an ENS name or an address.

The signature (*sig*) can be:

- A fragment: `someFunction(uint256,bytes32)`
- A selector and encoded calldata: `0xcdba2fd4000000000000000000000000000000000000000000000000000000000000007a69`
- Only the function name: in this case Cast will try to fetch the function signature from Etherscan

OPTIONS

Query Options

`-B` *block*
`--block` *block*
The block height you want to query at.

Can be a block number, or any of the tags: `earliest`, `finalized`, `safe`, `latest` or `pending`.

WALLET OPTIONS - RAW:

`-i`
`--interactive <NUM>`
Open an interactive prompt to enter your private key. Takes a value for the number of keys to enter.
Defaults to `0`.

`--mnemonic-derivation-path <PATHS>`
The wallet derivation path. Works with both `--mnemonic-path` and hardware wallets.

`--mnemonic-indexes <INDEXES>`
Use the private key from the given mnemonic index. Used with --mnemonic-paths.
Defaults to `0`.

`--mnemonic-passphrase <PASSPHRASE>`
Use a BIP39 passphrases for the mnemonic.

`--mnemonic <PATHS>`
Use the mnemonic phrases or mnemonic files at the specified paths.

`--private-key <RAW_PRIVATE_KEY>`
Use the provided private key.

`--private-keys <RAW_PRIVATE_KEYS>`
Use the provided private keys.

Wallet Options - Keystore

`--keystore` *path*
Use the keystore in the given folder or file.
Environment: `ETH_KEYSTORE`

`--account` *account-name*
Use a keystore from the default keystores folder (~/.foundry/keystores) by its filename.
Environment: `ETH_KEYSTORE_ACCOUNT`

`--interactive`

`--password` *password*
The keystore password. Used with `--keystore`. Environment: `ETH_PASSWORD`

Wallet Options - Hardware Wallet

`-t`
`--trezor`

Use a Trezor hardware wallet.

`-l`
`--ledger`
Use a Ledger hardware wallet.

Wallet Options - Remote

`-f` *address*
`--from` *address*
Sign the transaction with the specified account on the RPC.
Environment: `ETH_FROM`

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

`--flashbots`
Use the Flashbots RPC URL (https://rpc.flashbots.net).

Etherscan Options

`--chain` *chain_name*
The Etherscan chain.

`--etherscan-api-key` *key*
Etherscan API key, or the key of an Etherscan configuration table.
Environment: `ETHERSCAN_API_KEY`

Common Options

`-h`
`--help`
Prints help information.

SEE ALSO

cast, cast send, cast publish, cast call

cast logs

NAME

cast logs - Get logs by signature or topic.

SYNOPSIS

`cast logs` [*options*] *sig_or_topic* [*topics_or_args...*]

DESCRIPTION

Get logs by signature or topic.

The (*sig_or_topic*) may either be the event signature or its hashed topic (located at topics[0]).

If using a signature, remaining arguments must be in their ordinary form. If using a topic, the arguments must be as they themselves appear as topics.

OPTIONS

Query Options

`--from-block` *from_block* The block height to start query at.

Can also be the tags: `earliest`, `finalized`, `safe`, `latest`, or `pending`.

`--to-block` *to_block* The block height to stop query at.

Can also be the tags: `earliest`, `finalized`, `safe`, `latest`, or `pending`.

`--address` *address* The contract address to filter on

WALLET OPTIONS - RAW:

`-i`
`--interactive <NUM>`
Open an interactive prompt to enter your private key. Takes a value for the number of

keys to enter.
Defaults to `0`.

`--mnemonic-derivation-path <PATHS>`
The wallet derivation path. Works with both `--mnemonic-path` and hardware wallets.

`--mnemonic-indexes <INDEXES>`
Use the private key from the given mnemonic index. Used with --mnemonic-paths.
Defaults to `0`.

`--mnemonic-passphrase <PASSPHRASE>`
Use a BIP39 passphrases for the mnemonic.

`--mnemonic <PATHS>`
Use the mnemonic phrases or mnemonic files at the specified paths.

`--private-key <RAW_PRIVATE_KEY>`
Use the provided private key.

`--private-keys <RAW_PRIVATE_KEYS>`
Use the provided private keys.

Wallet Options - Keystore

`--keystore` *path*
Use the keystore in the given folder or file.
Environment: `ETH_KEYSTORE`

`--account` *account-name*
Use a keystore from the default keystores folder (~/.foundry/keystores) by its filename.
Environment: `ETH_KEYSTORE_ACCOUNT`

`--interactive`

`--password` *password*
The keystore password. Used with `--keystore`. Environment: `ETH_PASSWORD`

Wallet Options - Hardware Wallet

`-t`
`--trezor`
Use a Trezor hardware wallet.

`-l`
`--ledger`
Use a Ledger hardware wallet.

Wallet Options - Remote

`-f` *address*
`--from` *address*
Sign the transaction with the specified account on the RPC.
Environment: `ETH_FROM`

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

`--flashbots`
Use the Flashbots RPC URL (https://rpc.flashbots.net).

Etherscan Options

`--chain` *chain_name*
The Etherscan chain.

`--etherscan-api-key` *key*
Etherscan API key, or the key of an Etherscan configuration table.
Environment: `ETHERSCAN_API_KEY`

EXAMPLES

1. Get logs using a signature:

```
cast logs --from-block 15537393 --to-block latest 'Transfer (address
indexed from, address indexed to, uint256 value)'
0x2e8ABfE042886E4938201101A63730D04F160A82
```

2. Get logs using a topic:

```
cast logs --from-block 15537393 --to-block latest
0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef
0x000000000000000000000000002e8abfe042886e4938201101a63730d04f160a82
```

SEE ALSO

cast

Block Commands

- cast find-block
- cast gas-price
- cast block-number
- cast basefee
- cast block
- cast age

cast find-block

NAME

cast-find-block - Get the block number closest to the provided timestamp.

SYNOPSIS

`cast find-block` [*options*] *timestamp*

DESCRIPTION

Get the block number closest to the provided timestamp.

OPTIONS

RPC Options

`--rpc-url` *url*
: The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Get the block number closest to New Years 2021

```
cast find-block 1609459200
```

SEE ALSO

cast

cast gas-price

NAME

cast-gas-price - Get the current gas price.

SYNOPSIS

`cast gas-price` [*options*]

DESCRIPTION

Get the current gas price.

OPTIONS

RPC Options

`--rpc-url` *url*
: The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Get the current gas price:

```
cast gas-price
```

SEE ALSO

cast, cast basefee

cast block-number

NAME

cast-block-number - Get the latest block number.

SYNOPSIS

`cast block-number` [*options*]

DESCRIPTION

Get the latest block number.

OPTIONS

RPC Options

`--rpc-url` *url*
: The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Get the latest block number:

```
cast block-number
```

SEE ALSO

cast, cast block

cast basefee

NAME

cast-base-fee - Get the basefee of a block.

SYNOPSIS

`cast base-fee` [*options*] [*block*]

DESCRIPTION

Get the basefee of a block.

The specified *block* can be a block number, or any of the tags: `earliest`, `finalized`, `safe`, `latest` or `pending`. Default to `latest`.

OPTIONS

RPC Options

`--rpc-url` *url*
: The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Get the basefee of the latest block:

```
cast base-fee
```

2. Get the basefee of the genesis block:

```
cast base-fee 1
```

SEE ALSO

cast, cast block, cast age

cast block

NAME

cast-block - Get information about a block.

SYNOPSIS

`cast block` [*options*] [*block*]

DESCRIPTION

Get information about a block.

The specified *block* can be a block number, or any of the tags: `earliest`, `finalized`, `safe`, `latest` or `pending`. Default to `latest`.

OPTIONS

`-f` *field*
`--field` *field*
If specified, only get the given field of the block.

Display Options

`-j`
`--json`
Print the deployment information as JSON.

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Get the latest block:

```
cast block
```

2. Get the `finalized` block:

```
cast block finalized
```

3. Get the hash of the latest block:

```
cast block latest -f hash
```

SEE ALSO

cast, cast basefee, cast age

cast age

NAME

cast-age - Get the timestamp of a block.

SYNOPSIS

`cast age` [*options*] [*block*]

DESCRIPTION

Get the timestamp of a block.

The specified *block* can be a block number, or any of the tags: `earliest`, `finalized`, `safe`, `latest` or `pending`. Default to `latest`.

OPTIONS

RPC Options

`--rpc-url` *url*
: The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Get the timestamp of the latest block:

```
cast age
```

2. Get the timestamp of the genesis block:

```
cast age 1
```

SEE ALSO

cast, cast block, cast basefee

Account Commands

- cast balance
- cast storage
- cast proof
- cast nonce
- cast code
- cast codesize

cast balance

NAME

cast-balance - Get the balance of an account in wei.

SYNOPSIS

`cast balance` [*options*] *who*

DESCRIPTION

Get the balance of an account.

The argument *who* can be an ENS name or an address.

OPTIONS

Query Options

`-B` *block*
`--block` *block*
The block height you want to query at.

Can be a block number, or any of the tags: `earliest`, `finalized`, `safe`, `latest` or `pending`.

`-e` *ether*
`--ether` *ether*
If this flag is used then balance will be shown in ether

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Get the balance of beer.eth

```
cast balance beer.eth
```

2. Get the ERC20 balance of any address using RPC URL

```
# To load the variables in the .env file
source .env

# To get the USDT balance of Binance
cast balance --erc20 0xdAC17F958D2ee523a2206206994597C13D831ec7
0xF977814e90dA44bFA03b6295A0616a897441aceC --rpc-url $MAINNET_RPC_URL
```

SEE ALSO

cast, cast nonce

cast storage

NAME

cast-storage - Get the raw value of a contract's storage slot or its full storage layout.

SYNOPSIS

`cast storage` [*options*] *address slot*

DESCRIPTION

Get the raw value of a contract's storage slot. (Slot locations greater than 18446744073709551615 (u64::MAX) should be given as hex. Use `cast index` to compute mapping slots.)

Emit the slot number to get the full storage layout (requires contract to be verified on Etherscan with a Solidity version > 0.6.5 or you must be in a Forge project with a local contract matching the deployed bytecode).

The address (*address*) can be an ENS name or an address.

OPTIONS

Query Options

`-B` *block* `--block` *block* The block height you want to query at.

Can be a block number, or any of the tags: `earliest`, `finalized`, `safe`, `latest` or `pending`.

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Get the value of slot 0 on the WETH contract.

```
cast storage 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2 0
```

2. Get the full storage layout of the Milady contract.

```
cast storage 0x5Af0D9827E0c53E4799BB226655A1de152A425a5
```

SEE ALSO

cast, cast proof

cast proof

NAME

cast-proof - Generate a storage proof for a given storage slot.

SYNOPSIS

`cast proof` [*options*] *address* [*slots...*]

DESCRIPTION

Generate a storage proof for a given storage slot.

The address (*address*) can be an ENS name or an address.

The displayed output is a JSON object with the following keys:

- `accountProof` : Proof for the account itself
- `address` : The address of the account
- `balance` : The balance of the account
- `codeHash` : A hash of the account's code
- `nonce` : The nonce of the account
- `storageHash` : A hash of the account's storage
- `storageProof` : An array of storage proofs, one for each requested slot
- `storageProof.key` : The slot
- `storageProof.proof` : The proof for the slot
- `storageProof.value` : The value of the slot

OPTIONS

Query Options

`-B` *block*
`--block` *block*
The block height you want to query at.

Can be a block number, or any of the tags: `earliest`, `finalized`, `safe`, `latest` or `pending`.

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Display Options

`-j`
`--json`
Print the deployment information as JSON.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Get the proof for storage slot 0 on the WETH contract:

```
cast proof 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2 0
```

SEE ALSO

cast, cast storage

cast nonce

NAME

cast-nonce - Get the nonce for an account.

SYNOPSIS

`cast nonce` [*options*] *who*

DESCRIPTION

Get the nonce of an account.

The argument *who* can be an ENS name or an address.

OPTIONS

Query Options

`-B` *block*
`--block` *block*
The block height you want to query at.

Can be a block number, or any of the tags: `earliest`, `finalized`, `safe`, `latest` or `pending`.

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`

Prints help information.

EXAMPLES

1. Get the nonce of beer.eth

```
cast nonce beer.eth
```

SEE ALSO

cast, cast balance

cast code

NAME

cast-code - Get the bytecode of a contract.

SYNOPSIS

`cast code` [*options*] *address*

DESCRIPTION

Get the bytecode of a contract.

The contract (*address*) can be an ENS name or an address.

OPTIONS

Query Options

`-B` *block*
`--block` *block*
The block height you want to query at.

Can be a block number, or any of the tags: `earliest`, `finalized`, `safe`, `latest` or `pending`.

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`

Prints help information.

EXAMPLES

1. Get the bytecode of the WETH contract.

```
cast code 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
```

SEE ALSO

cast, cast proof

cast codesize

NAME

cast-codesize - Get the runtime bytecode size of a contract.

SYNOPSIS

`cast codesize` [*options*] *address*

DESCRIPTION

Get the runtime bytecode size of a contract.

The contract (*address*) can be an ENS name or an address.

OPTIONS

Query Options

`-B` *block*
`--block` *block*
: The block height you want to query at.

Can be a block number, or any of the tags: `earliest`, `finalized`, `safe`, `latest` or `pending`.

RPC Options

`--rpc-url` *url*
: The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`

Prints help information.

EXAMPLES

1. Get the runtime bytecode size of the WETH contract.

```
cast codesize 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
```

SEE ALSO

cast, cast code

ENS Commands

- cast lookup-address
- cast resolve-name
- cast namehash

cast lookup-address

NAME

cast-lookup-address - Perform an ENS reverse lookup.

SYNOPSIS

`cast lookup-address` [*options*] *who*

DESCRIPTION

Perform an ENS reverse lookup.

If `--verify` is passed, then a normal lookup is performed after the reverse lookup to verify that the address is correct.

OPTIONS

Lookup Options

`-v`
`--verify`
: Perform a normal lookup to verify that the address is correct.

RPC Options

`--rpc-url` *url*
: The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Get the ENS name for an address.

```
cast lookup-address $ADDRESS
```

2. Perform both a reverse and a normal lookup:

```
cast lookup-address --verify $ADDRESS
```

SEE ALSO

cast, cast resolve-name

cast resolve-name

NAME

cast-resolve-name - Perform an ENS lookup.

SYNOPSIS

`cast resolve-name` [*options*] *who*

DESCRIPTION

Perform an ENS lookup.

If `--verify` is passed, then a reverse lookup is performed after the normal lookup to verify that the name is correct.

OPTIONS

Lookup Options

`-v`
`--verify`
: Perform a reverse lookup to verify that the name is correct.

RPC Options

`--rpc-url` *url*
: The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Get the address for an ENS name.

```
cast resolve-name vitalik.eth
```

2. Perform both a normal and a reverse lookup:

```
cast resolve-name --verify vitalik.eth
```

SEE ALSO

cast, cast lookup-address

cast namehash

NAME

cast-namehash - Calculate the ENS namehash of a name.

SYNOPSIS

`cast namehash` [*options*] *name*

DESCRIPTION

Calculate the ENS namehash of a name.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Calculate the namehash of an ENS name.

```
cast namehash vitalik.eth
```

SEE ALSO

cast, cast lookup-address, cast resolve-name

Etherscan Commands

- cast etherscan-source

cast etherscan-source

NAME

cast-etherscan-source - Get the source code of a contract from Etherscan.

SYNOPSIS

`cast etherscan-source` [*options*] *address*

DESCRIPTION

Get the source code of a contract from Etherscan.

The destination (*to*) can be an ENS name or an address.

OPTIONS

Output Options

`-d` *directory*
The output directory to expand the source tree into. If not provided, the source will be outputted to stdout.

Etherscan Options

`--chain` *chain_name*
The Etherscan chain.

`--etherscan-api-key` *key*
Etherscan API key, or the key of an Etherscan configuration table.
Environment: `ETHERSCAN_API_KEY`

Common Options

`-h`
`--help`

Prints help information.

EXAMPLES

1. Get the source code of the WETH contract:

```
cast etherscan-source 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
```

2. Expand the source code of the WETH contract into a directory named `weth`

```
cast etherscan-source -d weth 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
```

SEE ALSO

cast

ABI Commands

- cast abi-decode
- cast abi-encode
- cast 4byte
- cast 4byte-decode
- cast 4byte-event
- cast calldata
- cast calldata-decode
- cast pretty-calldata
- cast selectors
- cast upload-signature

cast abi-decode

NAME

cast-abi-decode - Decode ABI-encoded input or output data.

SYNOPSIS

`cast abi-decode` [*options*] *sig calldata*

DESCRIPTION

Decode ABI-encoded input or output data.

By default, the command will decode output data. To decode input data, pass `--input` or use `cast calldata-decode`.

The signature (*sig*) is a fragment in the form `<function name>(<types...>)(<types...>)`.

OPTIONS

Decoder Options

`-i`
`--input`
Decode input data.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Decode output data for a `balanceOf` call:

```
cast abi-decode "balanceOf(address)(uint256)" \
  0x000000000000000000000000000000000000000000000000000000000000000a
```

2. Decode input data for a `transfer` call:

```
cast abi-decode --input "transfer(address,uint256)" \

0xa9059cbb000000000000000000000000e78388b4ce79068e89bf8aa7f218ef6b9ab0e9d0
000000000000000000000000000000000000000000000000008a8e4b1a3d8000
```

SEE ALSO

cast, cast calldata-decode

cast abi-encode

NAME

cast-abi-encode - ABI encode the given function arguments, excluding the selector.

SYNOPSIS

`cast abi-encode` [*options*] *sig* [*args...*]

DESCRIPTION

ABI encode the given function, excluding the selector.

The signature (*sig*) is a fragment in the form `<function name>(<types...>)`.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. ABI-encode the arguments for a call to `someFunc(address,uint256)`:

```
cast abi-encode "someFunc(address,uint256)" 0x... 1
```

2. For encoding a type with components (as a tuple, or custom struct):

```
cast abi-encode "someFunc((string,uint256))" "(myString,1)"
```

SEE ALSO

cast, cast calldata

cast 4byte

NAME

cast-4byte - Get the function signatures for the given selector from https://sig.eth.samczsun.com.

SYNOPSIS

`cast 4byte` [*options*] *sig*

DESCRIPTION

Get the function signatures for the given selector from https://sig.eth.samczsun.com.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Get the function signature for the selector `0x8cc5ce99`:

```
cast 4byte 0x8cc5ce99
```

SEE ALSO

cast, cast 4byte-decode, cast 4byte-event, cast selectors

cast 4byte-decode

NAME

cast-4byte-decode - Decode ABI-encoded calldata using https://sig.eth.samczsun.com.

SYNOPSIS

`cast 4byte-decode` [*options*] *calldata*

DESCRIPTION

Decode ABI-encoded calldata using https://sig.eth.samczsun.com.

OPTIONS

4byte Options

`--id` *id*
: The index of the resolved signature to use.
https://sig.eth.samczsun.com can have multiple possible signatures for a given selector.
The index can be an integer, or the tags “earliest” and “latest”.

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Decode calldata for a `transfer` call:

```
cast 4byte-decode
0xa9059cbb000000000000000000000000e78388b4ce79068e89bf8aa7f218ef6b9ab0e9d0
0000000000000000000000000000000000000000000000000174b37380cea000
```

SEE ALSO

cast, cast 4byte, cast 4byte-event

cast 4byte-event

NAME

cast-4byte-event - Get the event signature for a given topic 0 from https://sig.eth.samczsun.com.

SYNOPSIS

`cast 4byte-event` [*options*] *topic_0*

DESCRIPTION

Get the event signature for a given topic 0 from https://sig.eth.samczsun.com.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Get the event signature for a topic 0 of `0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef`:

```
cast 4byte-event
0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef
```

SEE ALSO

cast, cast 4byte, cast 4byte-decode

cast calldata

NAME

cast-calldata - ABI-encode a function with arguments.

SYNOPSIS

`cast calldata` [*options*] *sig* [*args...*]

DESCRIPTION

ABI-encode a function with arguments.

The signature (*sig*) is a fragment in the form `<function name>(<types...>)`.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. ABI-encode the arguments for a call to `someFunc(address,uint256)`:

```
cast calldata "someFunc(address,uint256)" 0x... 1
```

SEE ALSO

cast, cast abi-encode

cast calldata-decode

NAME

cast-calldata-decode - Decode ABI-encoded input data.

SYNOPSIS

`cast calldata-decode` [*options*] *sig calldata*

DESCRIPTION

Decode ABI-encoded input data.

The signature (*sig*) is a fragment in the form `<function name>(<types...>)`.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Decode input data for a `transfer` call:

```
cast calldata-decode "transfer(address,uint256)" \

0xa9059cbb000000000000000000000000e78388b4ce79068e89bf8aa7f218ef6b9ab0e9d0
0000000000000000000000000000000000000000000000008a8e4b1a3d8000
```

SEE ALSO

cast, cast abi-decode

cast pretty-calldata

NAME

cast-pretty-calldata - Pretty print calldata.

SYNOPSIS

`cast pretty-calldata` [*options*] *calldata*

DESCRIPTION

Pretty print calldata.

Tries to decode the calldata using https://sig.eth.samczsun.com unless `--offline` is passed.

OPTIONS

4byte Options

`-o`
`--offline`
: Skip the https://sig.eth.samczsun.com lookup.

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Decode calldata for a `transfer` call:

```
cast pretty-calldata
0xa9059cbb000000000000000000000000e78388b4ce79068e89bf8aa7f218ef6b9ab0e9d0
0000000000000000000000000000000000000000000000000174b37380cea000
```

SEE ALSO

cast, cast 4byte-decode

cast selectors

NAME

cast-selectors - Extracts function selectors and arguments from bytecode

SYNOPSIS

`cast selectors` [*options*] *bytecode*

DESCRIPTION

Extracts function selectors and arguments from bytecode using the EVMole library

OPTIONS

`-r`
`--resolve`
Resolve the function signatures for the extracted selectors using https://openchain.xyz

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Get WETH's contract function signatures & arguments:

```
cast selectors $(cast code 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2)
```

SEE ALSO

cast, cast 4byte

cast upload-signature

NAME

cast-upload-signature

SYNOPSIS

`cast upload-signature` [*signatures...*]

DESCRIPTION

Upload the given signatures to https://sig.eth.samczsun.com.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Upload signatures

```
cast upload-signature 'function approve(address,uint256)' \
'transfer(uint256)' 'event Transfer(uint256,address)'
```

Conversion Commands

- cast format-bytes32-string
- cast from-bin
- cast from-fixed-point
- cast from-rlp
- cast from-utf8
- cast from-wei
- cast parse-bytes32-address
- cast parse-bytes32-string
- cast to-ascii
- cast to-base
- cast to-bytes32
- cast to-dec
- cast to-fixed-point
- cast to-hex
- cast to-hexdata
- cast to-int256
- cast to-rlp
- cast to-uint256
- cast to-unit
- cast to-wei
- cast shl
- cast shr

cast format-bytes32-string

NAME

cast-format-bytes32-string - Formats a string into bytes32 encoding.

SYNOPSIS

`cast format-bytes32-string` [*options*] *string*

DESCRIPTION

Formats a string into bytes32 encoding.

Note that this command is for formatting a Solidity string literal into `bytes32` only. If you're looking to pad a byte string, use to-bytes32 instead.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Turn string "hello" into bytes32 hex:

```
cast format-bytes32-string "hello"
```

SEE ALSO

cast

cast from-bin

NAME

cast-from-bin - Convert binary data into hex data.

SYNOPSIS

`cast from-bin` [*options*]

DESCRIPTION

Convert binary data into hex data.

The input is taken from stdin.

OPTIONS

Common Options

`-h`
`--help`
: Prints help information.

SEE ALSO

cast

cast from-fixed-point

NAME

cast-from-fixed-point - Convert a fixed point number into an integer.

SYNOPSIS

`cast from-fixed-point` [*options*] *decimals value*

DESCRIPTION

Convert a fixed point number into an integer.

OPTIONS

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Convert 10.55 to an integer:

```
cast from-fixed-point 2 10.55
```

SEE ALSO

cast

cast from-rlp

NAME

cast-from-rlp - Decodes RLP-encoded data.

SYNOPSIS

`cast from-rlp` *data*

DESCRIPTION

Decodes RLP-encoded data.

The *data* is a hexadecimal string with optional 0x prefix.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Decode RLP data:

```
cast from-rlp 0xc481f181f2
```

```
cast from-rlp c481f181f2
```

cast from-utf8

NAME

cast-from-utf8 - Convert UTF8 text to hex.

SYNOPSIS

`cast from-utf8` [*options*] *text*

DESCRIPTION

Convert UTF8 text to hex.

OPTIONS

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Convert UTF8 text to hex:

```
cast from-utf8 "hello"
```

SEE ALSO

cast

cast from-wei

NAME

cast-from-wei - Convert wei into an ETH amount.

SYNOPSIS

`cast from-wei` [*options*] *value* [*unit*]

DESCRIPTION

Convert wei into an ETH amount.

Consider using `cast to-unit`.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

SEE ALSO

cast, cast calldata

cast parse-bytes32-address

NAME

cast-parse-bytes32-address - Parses a checksummed address from bytes32 encoding.

SYNOPSIS

`cast parse-bytes32-address` [*options*] *bytes*

DESCRIPTION

Parses a checksummed address from its bytes32 encoding representation.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Parse the bytes32 encoding of the WETH9 contract address to its address representation:

```
cast parse-bytes32-address
0x000000000000000000000000C02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
```

SEE ALSO

cast

cast parse-bytes32-string

NAME

cast-parse-bytes32-string - Parses a string from bytes32 encoding.

SYNOPSIS

`cast parse-bytes32-string` [*options*] *bytes*

DESCRIPTION

Parses a Solidity string literal from its bytes32 encoding representation mostly by interpreting bytes as ASCII characters. This command undos the encoding in –format-bytes32-string.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Parse bytes32 string encoding of “hello” back to the string representation:

```
cast parse-bytes32-string
"0x68656c6c6f000000000000000000000000000000000000000000000000000000"
```

SEE ALSO

cast

cast to-ascii

NAME

cast-to-ascii - Convert hex data to an ASCII string.

SYNOPSIS

`cast to-ascii` [*options*] *text*

DESCRIPTION

Convert hex data to an ASCII string.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Convert hex data to an ASCII string:

```
cast to-ascii "0x68656c6c6f"
```

SEE ALSO

cast

cast to-base

NAME

cast-to-base - Convert a number of one base to another.

SYNOPSIS

`cast to-base` [*options*] *value base*

DESCRIPTION

Convert a number of one base to another.

OPTIONS

Base Options

`--base-in` *base* The base of the input number. Available options:

10, d, dec, decimal

16, h, hex, hexadecimal

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Convert the decimal number 64 to hexadecimal

```
cast to-base 64 hex
```

2. Convert the hexadecimal number 100 to binary

```
cast to-base 0x100 2
```

Note: The –base-in parameter is not enforced but will be needed if the input is ambiguous.

SEE ALSO

cast

cast to-bytes32

NAME

cast-to-bytes32 - Right-pads hex data to 32 bytes.

SYNOPSIS

`cast to-bytes32` [*options*] *bytes*

DESCRIPTION

Right-pads hex data to 32 bytes.

Note that this command is for padding a byte string only. If you're looking to format a Solidity string literal into `bytes32`, use format-bytes32-string instead.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

SEE ALSO

cast

cast to-dec

NAME

cast-to-dec - Converts a number of one base to decimal

SYNOPSIS

`cast to-dec` [*options*] *value*

DESCRIPTION

Converts a number of one base to decimal

OPTIONS

`--base-in` *base_in* The input base.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Convert ff in hexadecimal to decimal

```
cast to-dec ff
```

SEE ALSO

cast

cast to-fixed-point

NAME

cast-to-fixed-point - Convert an integer into a fixed point number.

SYNOPSIS

`cast to-fixed-point` [*options*] *decimals value*

DESCRIPTION

Convert an integer into a fixed point number.

OPTIONS

Common Options

`-h`
`--help`
: Prints help information.

EXAMPLES

1. Convert 250 to a fixed point number with 2 decimals:

```
cast to-fixed-point 2 250
```

SEE ALSO

cast

cast to-hex

NAME

cast-to-hex - Converts a number of one base to another

SYNOPSIS

`cast to-hex` [*options*] *value*

DESCRIPTION

Converts a number of one base to another

OPTIONS

`--base-in` *base_in* The input base.

Common Options

`-h`
`--help`
Prints help information.

SEE ALSO

cast

cast to-hexdata

NAME

cast-to-hexdata - Normalize the input to lowercase, 0x-prefixed hex.

SYNOPSIS

`cast to-hexdata` [*options*] *input*

DESCRIPTION

Normalize the input to lowercase, 0x-prefixed hex.

The input data (*input*) can either be:

- Mixed case hex with or without the 0x prefix.
- 0x prefixed hex that should be concatenated, separated by `:`.
- An absolute path to a file containing hex.
- A `@tag`, where the tag is defined in an environment variable.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Add 0x prefix:

```
cast to-hexdata deadbeef
```

2. Concatenate hex values:

```
cast to-hexdata "deadbeef:0xbeef"
```

3. Normalize hex value in `MY_VAR`:

```
cast to-hexdata "@MY_VAR"
```

SEE ALSO

cast

cast to-int256

NAME

cast-to-int256 - Convert a number to a hex-encoded int256.

SYNOPSIS

`cast to-int256` [*options*] *value*

DESCRIPTION

Convert a number to a hex-encoded int256.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

SEE ALSO

cast

cast to-rlp

NAME

cast-to-rlp - Encodes hex data to RLP.

SYNOPSIS

`cast to-rlp` *array*

DESCRIPTION

RLP encodes a hex string or a JSON array of hex strings.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Encoding RLP data:

```
cast to-rlp '["0xaa","0xbb","cc"]'

cast to-rlp f0a9
```

cast to-uint256

NAME

cast-to-uint256 - Convert a number to a hex-encoded uint256.

SYNOPSIS

`cast to-uint256` [*options*] *value*

DESCRIPTION

Convert a number to a hex-encoded uint256.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

SEE ALSO

cast

cast to-unit

NAME

cast-to-unit - Convert an eth amount to another unit.

SYNOPSIS

`cast to-unit` [*options*] *value* [*unit*]

DESCRIPTION

Convert an eth amount to another unit.

The value to convert (*value*) can be a quantity of eth (in wei), or a number with a unit attached to it.

Valid units are:

- `ether`
- `gwei`
- `wei`

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Convert 1000 wei to gwei

```
cast to-unit 1000 gwei
```

2. Convert 1 eth to gwei

```
cast to-unit 1ether gwei
```

SEE ALSO

cast

cast to-wei

NAME

cast-to-wei - Convert an eth amount to wei.

SYNOPSIS

`cast to-wei` [*options*] *value* [*unit*]

DESCRIPTION

Convert an eth amount to wei.

Consider using `cast to-unit`.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

SEE ALSO

cast, cast calldata

cast shl

NAME

cast-shl - Perform a left shifting operation.

SYNOPSIS

`cast shl` [*options*] *value shift*

DESCRIPTION

Perform a left shifting operation.

OPTIONS

Base Options

`--base-in` *base* The base of the input number. Available options:

10, d, dec, decimal

16, h, hex, hexadecimal

`--base-out` *base* The desired base of the output. Available options:

2, b, bin, binary

8, o, oct, octal

10, d, dec, decimal

16, h, hex, hexadecimal

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Perform a 3 position left bit shift of the number 61

```
cast shl --base-in 10 61 3
```

Note: The –base-in parameter is not enforced but will be needed if the input is ambiguous.

SEE ALSO

cast, cast shr

cast shr

NAME

cast-shr - Perform a right shifting operation.

SYNOPSIS

`cast shr` [*options*] *value shift*

DESCRIPTION

Perform a right shifting operation.

OPTIONS

Base Options

`--base-in` *base* The base of the input number. Available options:

10, d, dec, decimal

16, h, hex, hexadecimal

`--base-out` *base* The desired base of the output. Available options:

2, b, bin, binary

8, o, oct, octal

10, d, dec, decimal

16, h, hex, hexadecimal

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Perform a single right bit shift of 0x12

```
cast shr --base-in 16 0x12 1
```

Note: The –base-in parameter is not enforced but will be needed if the input is ambiguous.

SEE ALSO

cast, cast shl

Utility Commands

- cast sig
- cast sig-event
- cast keccak
- cast compute-address
- cast create2
- cast interface
- cast index
- cast concat-hex
- cast max-int
- cast min-int
- cast max-uint
- cast to-check-sum-address

cast sig

NAME

cast-sig - Get the selector for a function.

SYNOPSIS

`cast sig` [*options*] *sig*

DESCRIPTION

Get the selector for a function.

The signature (*sig*) is a fragment in the form `<function name>(<types...>)`.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Get the selector for the function `transfer(address,uint256)`:

```
cast sig "transfer(address,uint256)"
```

2. Get the selector for a function that expects a `struct`:

```
contract Test {
    struct MyStruct {
        address addr;
        uint256 amount;
    }
    function myfunction(MyStruct memory t) public pure {}
}
```

Structs are encoded as tuples (see struct encoding).

```
cast sig "myfunction((address,uint256))"
```

SEE ALSO

cast, struct encoding

cast sig-event

NAME

cast-sig-event - Generate event signatures from event string.

SYNOPSIS

`cast sig-event` [*options*] *event_string*

DESCRIPTION

Generate event signatures from event string.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Get the hash for the log `Transfer(address indexed from, address indexed to, uint256 amount)`：

```
cast sig-event "Transfer(address indexed from, address indexed to, uint256 amount)"
```

SEE ALSO

cast

cast keccak

NAME

cast-keccak - Hash arbitrary data using keccak-256.

SYNOPSIS

`cast keccak` [*options*] *data*

DESCRIPTION

Hash arbitrary data using keccak-256.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

SEE ALSO

cast

cast compute-address

NAME

cast-compute-address - Compute the contract address from a given nonce and deployer address.

SYNOPSIS

`cast compute-address` [*options*] *address*

DESCRIPTION

Compute the contract address from a given nonce and deployer address.

OPTIONS

Compute Options

`--nonce` *nonce*
The nonce of the account. Defaults to the latest nonce, fetched from the RPC.

RPC Options

`--rpc-url` *url*
The RPC endpoint. Accepts a URL or an existing alias in the [rpc_endpoints] table, like `mainnet`. Environment: `ETH_RPC_URL`

Common Options

`-h`
`--help`
Prints help information.

SEE ALSO

cast, cast proof, cast create2

cast create2

NAME

cast-create2 - Generate a deterministic contract address using CREATE2

SYNOPSIS

`cast create2` [*options*]

DESCRIPTION

Generate a deterministic contract address using CREATE2

OPTIONS

`--starts-with` *hex* Prefix for the contract address.

`--ends-with` *hex* Suffix for the contract address

`--matching` *hex* Sequence that the address has to match

`--case-sensitive` Case sensitive matching

`--deployer` *address* Address of the contract deployer [default: `0x4e59b44847b379578588920ca78fbf26c0b4956c`]

`--init-code` *hex* Init code of the contract to be deployed

`--init-code-hash` *hash* Init code hash of the contract to be deployed

`--jobs` *jobs* Number of threads to use. Defaults to and caps at the number of logical cores

`--caller` *address* Address of the caller. Used for the first 20 bytes of the salt

Common Options

`-h`
`--help`

Prints help information.

EXAMPLES

1. Generate a contract address that starts with `dead`:

```
cast create2 --starts-with dead
```

2. Generate a contract address that ends with `beef`:

```
cast create2 --ends-with beef
```

3. A more complex example:

```
cast create2 --starts-with dead --case-sensitive --deployer
0x0000000000FFe8B47B3e2130213B802212439497 --init-code-hash
0x0c591f26891d6443cf08c5be3584c1e6ae10a4c2f07c5c53218741e9755fb9cd
```

SEE ALSO

cast, cast compute-address

cast interface

NAME

cast-interface - Generate a Solidity interface from a given ABI.

SYNOPSIS

`cast interface` [*options*] *address_or_path*

DESCRIPTION

Generates a Solidity interface from a given ABI.

The argument (*address_or_path*) can either be the path to a file containing an ABI, or an address.

If an address is provided, then the interface is generated from the ABI of the account, which is fetched from Etherscan.

> ℹ️ **Note**
>
> This command does not currently support ABI encoder v2.

OPTIONS

Interface Options

`-n` *name*
`--name` *name*
: The name to use for the generated interface. The default name is `Interface`.

`-o` *path*
: The path to the output file. If not specified, the interface will be output to stdout.

`-p` *version*
`--pragma` *version*
: The Solidity pragma version to use in the interface. Default: `^0.8.10`.

`-j`
`--json`
Output the contract's JSON ABI.

Etherscan Options

`--chain` *chain_name*
The Etherscan chain.

`--etherscan-api-key` *key*
Etherscan API key, or the key of an Etherscan configuration table.
Environment: `ETHERSCAN_API_KEY`

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Generate an interface from a file:

```
cast interface ./path/to/abi.json
```

2. Generate an interface using Etherscan:

```
cast interface -o IWETH.sol 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
```

3. Generate and name an interface from a file:

```
cast interface -n LilENS ./path/to/abi.json
```

4. Fetch the JSON ABI of a contract on Etherscan:

```
cast interface -o IWETH.sol -j 0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2
```

SEE ALSO

cast, cast proof

cast index

NAME

cast-index - Compute the storage slot location for an entry in a mapping.

SYNOPSIS

`cast index` *key_type key slot*

DESCRIPTION

Compute the storage slot location for an entry in a mapping.

Use `cast storage` to get the value.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

```
// World.sol

mapping (address => uint256) public mapping1;
mapping (string => string) public mapping2;
```

1. Compute the storage slot of an entry (`hello`) in a mapping of type `mapping(string => string)` , located at slot 1:

```
>> cast index string "hello" 1
0x3556fc8e3c702d4479a1ab7928dd05d87508462a12f53307b5407c969223d1f8
>> cast storage [address]
0x3556fc8e3c702d4479a1ab7928dd05d87508462a12f53307b5407c969223d1f8
world
```

SEE ALSO

cast

cast concat-hex

NAME

cast-concat-hex - Concatenate hex strings.

SYNOPSIS

`cast concat-hex` *data...*

DESCRIPTION

Concatenate hex strings.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Concatenate hex strings:

```
cast concat-hex 0xa 0xb 0xc
```

SEE ALSO

cast

cast max-int

NAME

cast-max-int - Get the maximum i256 value.

SYNOPSIS

```
cast max-int
```

DESCRIPTION

Get the maximum i256 value.

OPTIONS

Common Options

`-h`
`--help`
: Prints help information.

SEE ALSO

cast, cast min-int, cast max-uint

cast min-int

NAME

cast-min-int - Get the minimum i256 value.

SYNOPSIS

```
cast min-int
```

DESCRIPTION

Get the minimum i256 value.

OPTIONS

Common Options

`-h`
`--help`
: Prints help information.

SEE ALSO

cast, cast max-int

cast max-uint

NAME

cast-max-uint - Get the maximum uint256 value.

SYNOPSIS

```
cast max-uint
```

DESCRIPTION

Get the maximum uint256 value.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

SEE ALSO

cast, cast max-int

cast to-check-sum-address

NAME

cast-to-check-sum-address - Convert an address to a checksummed format (EIP-55).

SYNOPSIS

`cast to-check-sum-address` *address*

DESCRIPTION

Convert an address to a checksummed format (EIP-55).

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

SEE ALSO

cast

Wallet Commands

- cast wallet
- cast wallet address
- cast wallet new
- cast wallet sign
- cast wallet vanity
- cast wallet verify
- cast wallet import

cast wallet

NAME

cast-wallet - Wallet management utilities.

SYNOPSIS

`cast wallet` [*options*] *command* [*args*]
`cast wallet` [*options*] `--version`
`cast wallet` [*options*] `--help`

DESCRIPTION

This program is a set of tools to use, create and manage wallets.

COMMANDS

cast wallet new
: Create a new random keypair.

cast wallet address
: Convert a private key to an address.

cast wallet sign
: Sign a message.

cast wallet vanity
: Generate a vanity address.

cast wallet verify
: Verify the signature of a message.

cast wallet import
: Import a private key into an encrypted keystore.

cast wallet list
: List all the accounts in the keystore default directory.

OPTIONS

Special Options

`-V`
`--version`
: Print version info and exit.

Common Options

`-h`
`--help`
: Prints help information.

cast wallet new

NAME

cast-wallet-new - Create a new random keypair.

SYNOPSIS

`cast wallet new` [*options*] [*path*]

DESCRIPTION

Create a new random keypair.

If *path* is specified, then the new keypair will be written to a JSON keystore encrypted with a password. (*path* should be an existing directory.)

OPTIONS

Keystore Options

`-p`
`--password`
Triggers a hidden password prompt for the JSON keystore.
Deprecated: prompting for a hidden password is now the default.

`--unsafe-password` *password*
Password for the JSON keystore in cleartext.

This is **unsafe** to use and we recommend using `--password` instead.
Environment: `CAST_PASSWORD`

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Create a new keypair without saving it to a keystore:

```
cast wallet new
```

2. Create a new keypair and save it in the `keystore` directory:

```
cast wallet new keystore
```

SEE ALSO

cast, cast wallet

cast new-mnemonic

NAME

cast-wallet-new-mnemonic - Creates a new mnemonic with a set number of words.

SYNOPSIS

`cast wallet new-mnemonic` [*options*]

DESCRIPTION

Generates a random BIP39 mnemonic phrase.

OPTIONS

New Mnemonic Options

`-w --words` The amount of words for the mnemonic. Defaults to 12.

`-a --accounts` The number of accounts to display, which are derived from the mnemonic. Defaults to 1.

EXAMPLES

1. Create a new mnemonic with 24 words.

```
cast wallet new-mnemonic --words 24
```

```
Successfully generated a new mnemonic.
Phrase:
decrease where seek crop segment want icon medal sleep social blast provide
virus grief pledge soccer stereo trick dry dirt rotate explain into nominee

Accounts:
- Account 0:
Address:     0x34644D4eC92ae1832877cE22AD9bA4b00c7397c8
Private key: 0x832a3784d0a130c8a0ce3cc6dfc336a41ca7801a117eac7a3bfaace52e4d239c
```

SEE ALSO

cast

cast wallet address

NAME

cast-wallet-address - Convert a private key to an address.

SYNOPSIS

`cast wallet address` [*options*]

DESCRIPTION

Convert a private key to an address.

OPTIONS

Keystore Options

WALLET OPTIONS - RAW:

`-i`
`--interactive <NUM>`
Open an interactive prompt to enter your private key. Takes a value for the number of keys to enter.
Defaults to `0`.

`--mnemonic-derivation-path <PATHS>`
The wallet derivation path. Works with both `--mnemonic-path` and hardware wallets.

`--mnemonic-indexes <INDEXES>`
Use the private key from the given mnemonic index. Used with --mnemonic-paths.
Defaults to `0`.

`--mnemonic-passphrase <PASSPHRASE>`
Use a BIP39 passphrases for the mnemonic.

`--mnemonic <PATHS>`
Use the mnemonic phrases or mnemonic files at the specified paths.

`--private-key <RAW_PRIVATE_KEY>`
Use the provided private key.

`--private-keys <RAW_PRIVATE_KEYS>`
Use the provided private keys.

Wallet Options - Keystore

`--keystore` *path*
Use the keystore in the given folder or file.
Environment: `ETH_KEYSTORE`

`--account` *account-name*
Use a keystore from the default keystores folder (~/.foundry/keystores) by its filename.
Environment: `ETH_KEYSTORE_ACCOUNT`

`--interactive`

`--password` *password*
The keystore password. Used with `--keystore`. Environment: `ETH_PASSWORD`

Wallet Options - Hardware Wallet

`-t`
`--trezor`
Use a Trezor hardware wallet.

`-l`
`--ledger`
Use a Ledger hardware wallet.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Get the address of the keypair in `keystore.json`:

```
cast wallet address --keystore keystore.json
```

SEE ALSO

cast, cast wallet

cast wallet sign

NAME

cast-wallet-sign - Sign a message.

SYNOPSIS

`cast wallet sign` [*options*] *message*

DESCRIPTION

Sign a message.

OPTIONS

WALLET OPTIONS - RAW:

`-i`
`--interactive <NUM>`
Open an interactive prompt to enter your private key. Takes a value for the number of keys to enter.
Defaults to `0`.

`--mnemonic-derivation-path <PATHS>`
The wallet derivation path. Works with both `--mnemonic-path` and hardware wallets.

`--mnemonic-indexes <INDEXES>`
Use the private key from the given mnemonic index. Used with --mnemonic-paths.
Defaults to `0`.

`--mnemonic-passphrase <PASSPHRASE>`
Use a BIP39 passphrases for the mnemonic.

`--mnemonic <PATHS>`
Use the mnemonic phrases or mnemonic files at the specified paths.

`--private-key <RAW_PRIVATE_KEY>`
Use the provided private key.

`--private-keys <RAW_PRIVATE_KEYS>`
Use the provided private keys.

Wallet Options - Keystore

`--keystore` *path*
Use the keystore in the given folder or file.
Environment: `ETH_KEYSTORE`

`--account` *account-name*
Use a keystore from the default keystores folder (~/.foundry/keystores) by its filename.
Environment: `ETH_KEYSTORE_ACCOUNT`

`--interactive`

`--password` *password*
The keystore password. Used with `--keystore`. Environment: `ETH_PASSWORD`

Wallet Options - Hardware Wallet

`-t`
`--trezor`
Use a Trezor hardware wallet.

`-l`
`--ledger`
Use a Ledger hardware wallet.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Sign a message using a keystore:

```
cast wallet sign --keystore keystore.json --interactive "hello"
```

2. Sign a message using a raw private key:

```
cast wallet sign --private-key $PRIV_KEY "hello"
```

SEE ALSO

cast, cast wallet

cast wallet vanity

NAME

cast-wallet-vanity - Generate a vanity address.

SYNOPSIS

`cast wallet vanity` [*options*]

DESCRIPTION

Generate a vanity address.

If `--nonce` is specified, then the command will try to generate a vanity contract address. The `--save-path` option allows specifying a custom file path to save the generated wallet details.

OPTIONS

Keystore Options

`--starts-with` *hex*
Prefix for the vanity address.

`--ends-with` *hex*
Suffix for the vanity address.

`--nonce` *nonce*
Generate a vanity contract address created by the generated keypair with the specified nonce.

`--save-path` *path*
Path to save the generated vanity wallet. If provided, the wallet details will be saved in a JSON file at this location.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Create a new keypair that starts with `dead`:

```
cast wallet vanity --starts-with dead
```

2. Create a new keypair ends with `beef`:

```
cast wallet vanity --ends-with beef
```

3. Create a new keypair that starts with `dead` and save the details to a specific path:

```
cast wallet vanity --starts-with dead --save-path /path/to/save
```

SEE ALSO

cast, cast wallet

cast wallet verify

NAME

cast-wallet-verify - Verify the signature of a message.

SYNOPSIS

`cast wallet verify` [*options*] `--address` *address message signature*

DESCRIPTION

Verify the signature of a message.

OPTIONS

Signature Options

`-a` *address*
`--address` *address*
: The address of the message signer.

Common Options

`-h`
`--help`
: Prints help information.

SEE ALSO

cast, cast wallet

cast wallet import

NAME

cast-wallet-import - Import a private key into an encrypted keystore

SYNOPSIS

`cast wallet import` [*options*] *account_name*

DESCRIPTION

Import a private key into an encrypted keystore.

If no *keystore-dir* is specified, it will be saved in the default `~/.foundry/keystores`, so it can be accessed through the `--account` option in methods like `forge script`, `cast send` or any other that requires a private key.

OPTIONS

Directory Options

`-k`
`--keystore-dir`

The path to store the encrypted keystore.
Defaults to `~/.foundry/keystores`.

WALLET OPTIONS - RAW:

`-i`
`--interactive <NUM>`
Open an interactive prompt to enter your private key. Takes a value for the number of keys to enter.
Defaults to `0`.

`--mnemonic-derivation-path <PATHS>`
The wallet derivation path. Works with both `--mnemonic-path` and hardware wallets.

`--mnemonic-indexes <INDEXES>`
Use the private key from the given mnemonic index. Used with --mnemonic-paths. Defaults to `0`.

`--mnemonic-passphrase <PASSPHRASE>`
Use a BIP39 passphrases for the mnemonic.

`--mnemonic <PATHS>`
Use the mnemonic phrases or mnemonic files at the specified paths.

`--private-key <RAW_PRIVATE_KEY>`
Use the provided private key.

`--private-keys <RAW_PRIVATE_KEYS>`
Use the provided private keys.

Common Options

`-h`
`--help`
Prints help information.

EXAMPLES

1. Create a keystore from a private key:

```
cast wallet import BOB --interactive
```

2. Create a keystore from a mnemonic:

```
cast wallet import ALICE --mnemonic "test test test test test test test test test test test test"
```

3. Create a keystore from a mnemonic with a specific mnemonic index:

```
cast wallet import ALICE --mnemonic "test test test test test test test test test test test test" --mnemonic-index 1
```

SEE ALSO

cast, cast wallet

cast wallet list

NAME

cast-wallet-list - List all the accounts in the keystore default directory.

SYNOPSIS

```
cast wallet list
```

DESCRIPTION

List all the accounts in the keystore default directory `~/.foundry/keystores`.

OPTIONS

Common Options

`-h`
`--help`
Prints help information.

SEE ALSO

cast, cast wallet

anvil

NAME

anvil - Create a local testnet node for deploying and testing smart contracts. It can also be used to fork other EVM compatible networks.

SYNOPSIS

`anvil` [*options*]

DESCRIPTION

Create a local testnet node for deploying and testing smart contracts. It can also be used to fork other EVM compatible networks.

This section covers an extensive list of information about Mining Modes, Supported Transport Layers, Supported RPC Methods, Anvil flags and their usages. You can run multiple flags at the same time.

Mining Modes

Mining modes refer to how frequent blocks are mined using Anvil. By default, it automatically generates a new block as soon as a transaction is submitted.

You can change this setting to interval mining if you will, which means that a new block will be generated in a given period of time selected by the user. If you want to go for this type of mining, you can do it by adding the `--block-time <block-time-in-seconds>` flag, like in the following example.

```
# Produces a new block every 10 seconds
anvil --block-time 10
```

There's also a third mining mode called never. In this case, it disables auto and interval mining, and mine on demand instead. You can do this by typing:

```
# Enables never mining mode
anvil --no-mining
```

Supported Transport Layers

HTTP and Websocket connections are supported. The server listens on port 8545 by default, but it can be changed by running the following command:

```
anvil --port <PORT>
```

Default CREATE2 Deployer

Anvil, when used without forking, includes the default CREATE2 deployer proxy at the address `0x4e59b44847b379578588920ca78fbf26c0b4956c`.

This allows you to test CREATE2 deployments locally without forking.

Supported RPC Methods

Standard Methods

The standard methods are based on this reference.

- `web3_clientVersion`
- `web3_sha3`
- `eth_chainId`
- `eth_networkId`
- `eth_gasPrice`
- `eth_accounts`
- `eth_blockNumber`
- `eth_getBalance`
- `eth_getStorageAt`
- `eth_getBlockByHash`
- `eth_getBlockByNumber`
- `eth_getTransactionCount`
- `eth_getBlockTransactionCountByHash`
- `eth_getBlockTransactionCountByNumber`

- `eth_getUncleCountByBlockHash`
- `eth_getUncleCountByBlockNumber`
- `eth_getCode`
- `eth_sign`
- `eth_signTypedData_v4`
- `eth_sendTransaction`
- `eth_sendRawTransaction`
- `eth_call`
- `eth_createAccessList`
- `eth_estimateGas`
- `eth_getTransactionByHash`
- `eth_getTransactionByBlockHashAndIndex`
- `eth_getTransactionByBlockNumberAndIndex`
- `eth_getTransactionReceipt`
- `eth_getUncleByBlockHashAndIndex`
- `eth_getUncleByBlockNumberAndIndex`
- `eth_getLogs`
- `eth_newFilter`
- `eth_getFilterChanges`
- `eth_newBlockFilter`
- `eth_newPendingTransactionFilter`
- `eth_getFilterLogs`
- `eth_uninstallFilter`
- `eth_getWork`
- `eth_subscribe`
- `eth_unsubscribe`

- `eth_syncing`
- `eth_submitWork`
- `eth_submitHashrate`
- `eth_feeHistory`
- `eth_getProof`
- `debug_traceTransaction` Use `anvil --steps-tracing` to get `structLogs`
- `debug_traceCall` Note that non-standard traces are not yet support. This means you can't pass any arguments to the `trace` parameter.
- `trace_transaction`
- `trace_block`

Custom Methods

The `anvil_*` namespace is an alias for `hardhat`. For more info, refer to the Hardhat documentation.

`anvil_impersonateAccount` Send transactions impersonating an externally owned account or contract.

`anvil_stopImpersonatingAccount` Stops impersonating an account or contract if previously set with `anvil_impersonateAccount`

`anvil_autoImpersonateAccount` Accepts `true` to enable auto impersonation of accounts, and `false` to disable it. When enabled, any transaction's sender will be automatically impersonated. Same as `anvil_impersonateAccount`.

`anvil_getAutomine` Returns true if automatic mining is enabled, and false if it is not

`anvil_mine` Mines a series of blocks

`anvil_dropTransaction` Removes transactions from the pool

`anvil_reset` Reset the fork to a fresh forked state, and optionally update the fork config

`anvil_setRpcUrl` Sets the backend RPC URL

`anvil_setBalance` Modifies the balance of an account

`anvil_setCode` Sets the code of a contract

`anvil_setNonce` Sets the nonce of an address

`anvil_setStorageAt` Writes a single slot of the account's storage

`anvil_setCoinbase` Sets the coinbase address

`anvil_setLoggingEnabled` Enable or disable logging

`anvil_setMinGasPrice` Set the minimum gas price for the node

`anvil_setNextBlockBaseFeePerGas` Sets the base fee of the next block

`anvil_setChainId` Sets the chain ID of the current EVM instance

`anvil_dumpState` Returns a hex string representing the complete state of the chain. Can be re-imported into a fresh/restarted instance of Anvil to reattain the same state.

`anvil_loadState` When given a hex string previously returned by `anvil_dumpState`, merges the contents into the current chain state. Will overwrite any colliding accounts/storage slots.

`anvil_nodeInfo` Retrieves the configuration params for the currently running Anvil node.

Special Methods

The special methods come from Ganache. You can take a look at the documentation here.

`evm_setAutomine` Enables or disables, based on the single boolean argument, the automatic mining of new blocks with each new transaction submitted to the network

`evm_setIntervalMining` Sets the mining behavior to interval with the given interval (seconds)

`evm_snapshot` Snapshot the state of the blockchain at the current block

`evm_revert` Revert the state of the blockchain to a previous snapshot. Takes a single parameter, which is the snapshot id to revert to

`evm_increaseTime` Jump forward in time by the given amount of time, in seconds

`evm_setNextBlockTimestamp` Similar to `evm_increaseTime` but takes the exact timestamp that you want in the next block

`anvil_setBlockTimestampInterval` Similar to `evm_increaseTime` but sets a block timestamp `interval`. The timestamp of the next block will be computed as `lastBlock_timestamp + interval`

`evm_setBlockGasLimit` Sets the block gas limit for the following blocks

`anvil_removeBlockTimestampInterval` Removes an `anvil_setBlockTimestampInterval` if it exists

`evm_mine` Mine a single block

`anvil_enableTraces` Turn on call traces for transactions that are returned to the user when they execute a transaction (instead of just txhash/receipt)

`eth_sendUnsignedTransaction` Execute a transaction regardless of signature status

For the next three methods, make sure to read Geth's documentation.

`txpool_status` Returns the number of transactions currently pending for inclusion in the next block(s), as well as the ones that are being scheduled for future execution only

`txpool_inspect` Returns a summary of all the transactions currently pending for inclusion in the next block(s), as well as the ones that are being scheduled for future execution only

`txpool_content` Returns the details of all transactions currently pending for inclusion in the next block(s), as well as the ones that are being scheduled for future execution only

Otterscan Methods

The `ots_*` namespace implements the Otterscan specification

`ots_getApiLevel` Used by Otterscan to check if it's connecting to a compatible node and display a friendly message if it is not.

`ots_getInternalOperations` Returns the internal ETH transfers inside a transaction.

`ots_hasCode` Check if a certain address contains a deployed code.

`ots_getTransactionError` Extract the transaction raw error output.

`ots_traceTransaction` Extract all variations of calls, contract creation and self-destructs and returns a call tree.

`ots_getBlockDetails` Tailor-made and expanded version of eth_getBlock* for block details page in Otterscan.

`ots_getBlockTransactions` Get paginated transactions for a certain block, And removes some verbose fields such logs.

`ots_searchTransactionsBefore` Gets paginated inbound/outbound transaction calls for a certain address, and before a given target block

`ots_searchTransactionsAfter` Gets paginated inbound/outbound transaction calls for a certain address, and after a given target block

`ots_getTransactionBySenderAndNonce` Gets the transaction hash for a certain sender address, given its nonce.

`ots_getContractCreator` Gets the transaction hash and the address which created a contract.

OPTIONS

General Options

`-a, --accounts <ACCOUNTS>` Set the number of accounts [default: 10]

`--auto-impersonate` Enable autoImpersonate on startup

`-b, --block-time <block-time>` Block time in seconds for interval mining

`--balance <BALANCE>` Set the balance of the accounts [default: 10000]

`--derivation-path <DERIVATION_PATH>` Set the derivation path of the child key to be derived [default: m/44'/60'/0'/0/]

`-h, --help` Print help information

`--hardfork <HARDFORK>` Choose the EVM hardfork to use Choose the hardfork by name, e.g. `shanghai`, `paris`, `london`, etc... [default: latest]

`--init <PATH>` Initialize the genesis block with the given `genesis.json` file.

`-m, --mnemonic <MNEMONIC>` BIP39 mnemonic phrase used for generating accounts

`--no-mining` Disable auto and interval mining, and mine on demand instead

`--order <ORDER>` How transactions are sorted in the mempool [default: fees]

`-p, --port <PORT>` Port number to listen on [default: 8545]

`--steps-tracing` Enable steps tracing used for debug calls returning geth-style traces [aliases: tracing]

`--ipc [<PATH>]` Starts an IPC endpoint at the given `PATH` argument or the default path: unix: `tmp/anvil.ipc`, windows: `\\.\pipe\anvil.ipc`

`--silent` Don't print anything on startup

`--timestamp <TIMESTAMP>` Set the timestamp of the genesis block

`-V, --version` Print version information

`--disable-default-create2-deployer` Disables deploying the default CREATE2 factory when running Anvil without forking

EVM Options

`-f, --fork-url <URL>` Fetch state over a remote endpoint instead of starting from an empty state

`--fork-block-number <BLOCK>` Fetch state from a specific block number over a remote endpoint (Must pass –fork-url in the same command-line)

`--fork-retry-backoff <BACKOFF>` Initial retry backoff on encountering errors.

`--retries <retries>` Number of retry requests for spurious networks (timed out requests). [default value= 5]

`--timeout <timeout>` Timeout in ms for requests sent to remote JSON-RPC server in forking mode. [default value= 45000]

`--compute-units-per-second <CUPS>` Sets the number of assumed available compute units per second for this provider [default value=330] See also, Alchemy Ratelimits

`--no-rate-limit` Disables rate limiting for this node's provider. Will always override `--compute-units-per-second` if present. [default value= false] See also, Alchemy Ratelimits

`--no-storage-caching>` Explicitly disables the use of RPC caching. All storage slots are read entirely from the endpoint. This flag overrides the project's configuration file (Must pass –fork-url in the same command-line)

Executor Environment Config

`--base-fee <FEE> --block-base-fee-per-gas <FEE>` The base fee in a block

`--chain-id <CHAIN_ID>` The chain ID [default: 31337]

`--code-size-limit <CODE_SIZE>` EIP-170: Contract code size limit in bytes. Useful to increase this because of tests. By default, it is 0x6000 (~25kb)

`--gas-limit <GAS_LIMIT>` The block gas limit

`--gas-price <GAS_PRICE>` The gas price

Server Options

`--allow-origin <allow-origin>` Set the CORS allow_origin [default: *]

`--no-cors` Disable CORS

`--host <HOST>` The IP address the server will listen on

`--config-out <OUT_FILE>` Writes output of `anvil` as json to user-specified file

`--prune-history`	Don't keep full chain history

EXAMPLES

1. Set the number of accounts to 15 and their balance to 300 ETH

```
anvil --accounts 15 --balance 300
```

2. Choose the address which will execute the tests

```
anvil --sender 0xC8479C45EE87E0B437c09d3b8FE8ED14ccDa825E
```

3. Change how transactions are sorted in the mempool to FIFO

```
anvil --order fifo
```

Shell Completions

`anvil completions` *shell*

Generates a shell completions script for the given shell.

Supported shells are:

- bash
- elvish
- fish
- powershell
- zsh

EXAMPLES

1. Generate shell completions script for zsh:

```
anvil completions zsh > $HOME/.oh-my-zsh/completions/_anvil
```

Usage within Docker

In order to run anvil as a service in Github Actions with the Docker container, where passing arguments to the entrypoint command is not possible, use the `ANVIL_IP_ADDR` environment variable to set the host's IP. `ANVIL_IP_ADDR=0.0.0.0` is equivalent to providing the `--host <ip>` option.

Using `genesis.json`

The `genesis.json` file in Anvil serves a similar purpose as in Geth, defining the network's initial state, consensus rules, and preallocated accounts to ensure all nodes start consistently and maintain network integrity. All values, including balance, gas limit and such, are to be defined as hexadecimals.

- `chainId` : Identifier for the blockchain, unique to each network.
- `nonce` : A counter used in hashing algorithms to ensure data integrity.
- `timestamp` : The creation time of the genesis block in Unix time.
- `extraData` : Additional data that can be included by the creator of the genesis block.
- `gasLimit` : The maximum amount of gas that can be used in the block.
- `difficulty` : A measure of how difficult it is to mine a new block.
- `mixHash` : A unique identifier proving a sufficient amount of computation for the block.
- `coinbase` : The Ethereum address of the miner who mined this block.
- `stateRoot` : The root of the state trie, reflecting the final state after all transactions.
- `alloc` : Allows pre-allocating Ether to a set of addresses with predefined balances.
- `number` : The block number, with the genesis block being 0.
- `gasUsed` : The total gas used in the block.
- `parentHash` : The hash of the parent block, all zeros for the genesis block since there is no parent.

A sample for simulating mainnet via genesis can be found here.

```
{
  "chainId": "0x2323",
  "nonce": "0x42",
  "timestamp": "0x0",
  "extraData":
"0x11bbe8db4e347b4e8c937c1c8370e4b5ed33adb3db69cbdb7a38e1e50b1b82fa",
  "gasLimit": "0x1388",
  "difficulty": "0x400000000",
  "mixHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
  "coinbase": "0x0000000000000000000000000000000000000000",
  "stateRoot":
"0xd7f8974fb5ac78d9ac099b9ad5018bedc2ce0a72dad1827a1709da30580f0544",
  "alloc": {
    "000d836201318ec6899a67540690382780743280": {
      "balance": "0xad78ebc5ac6200000"
    }
  },
  "number": "0x0",
  "gasUsed": "0x0",
  "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000"
}
```

chisel

NAME

`chisel` - Test and receive verbose feedback on Solidity inputs within a REPL environment.

SYNOPSIS

`chisel` [*options*]

Subcommands (bin)

1. `chisel list`
   - Displays all cached sessions stored in `~/.foundry/cache/chisel`.
2. `chisel load <id>`
   - If a cached session with `id = <id>` exists, launches the REPL and loads the corresponding session.
3. `chisel view <id>`
   - If a cached session with `id = <id>` exists, displays the source code of the session's REPL contract.
4. `chisel clear-cache`
   - Deletes all cache files within the `~/.foundry/cache/chisel` directory. These sessions are unrecoverable, so use this command with care.

Flags

See `man chisel` or `chisel --help` for all available environment configuration flags.

DESCRIPTION

Chisel is a Solidity REPL (short for "read-eval-print loop") that allows developers to write and test Solidity code snippets. It provides an interactive environment for writing and executing Solidity code, as well as a set of built-in commands for working with and debugging your code. This makes it a useful tool for quickly testing and experimenting with Solidity code without having to spin up a sandbox foundry test suite.

Usage

To open chisel, simply execute the `chisel` binary.

From there, input valid Solidity code. There are two kinds of inputs to the chisel prompt apart from commands:

1. Expressions
   - Expressions are statements that return a value or otherwise can be evaluated on their own. For example, `1 << 8` is an expression that will evaluate to a `uint256` with the value `256`. Expressions will be evaluated up front, and will not persist in the session state past their evaluation.
   - Examples:
     - `address(0).balance`
     - `abi.encode(256, bytes32(0), "Chisel!")`
     - `myViewFunc(128)`
     - ...
2. Statements
   - Statements are snippets of code that are meant to persist in the session's state. Statements include variable definitions, calls to non-state-mutating functions that return a value, and contract, function, event, error, mapping, or struct definitions. If you would like an expression to be evaluated as a statement, a semi-colon (`;`) can be appended to the end.
   - Examples:
     - `uint256 a = 0xa57b`
     - `myStateMutatingFunc(128) || myViewFunc(128);` <- Notice the `;`
     -
```
function hash64(
  bytes32 _a,
  bytes32 _b
) internal pure returns (bytes32 _hash) {
    assembly {
        // Store the 64 bytes we want to hash in scratch space
        mstore(0x00, _a)
        mstore(0x20, _b)

        // Hash the memory in scratch space
        // and assign the result to `_hash`
        _hash := keccak256(0x00, 0x40)
    }
}
```
     - `event ItHappened(bytes32 indexed hash)`

- `struct Complex256 { uint256 re; uint256 im; }`
- ...

Available Commands

```
No solidity versions installed! Installing solidity version 0.8.19...
Welcome to Chisel! Type `!help` to show available commands.
⚒ Chisel help
=============
General
    !help | !h - Display all commands
    !quit | !q - Quit Chisel
    !exec <command> [args] | !e <command> [args] - Execute a shell command and
print the output

Session
    !clear | !c - Clear current session source
    !source | !so - Display the source code of the current session
    !save [id] | !s [id] - Save the current session to cache
    !load <id> | !l <id> - Load a previous session ID from cache
    !list | !ls - List all cached sessions
    !clearcache | !cc - Clear the chisel cache of all stored sessions
    !export | !ex - Export the current session source to a script file
    !fetch <addr> <name> | !fe <addr> <name> - Fetch the interface of a
verified contract on Etherscan
    !edit - Open the current session in an editor

Environment
    !fork <url> | !f <url> - Fork an RPC for the current session. Supply 0
arguments to return to a local network
    !traces | !t - Enable / disable traces for the current session
    !calldata [data] | !cd [data] - Set calldata (`msg.data`) for the current
session (appended after function selector). Clears it if no argument provided.

Debug
    !memdump | !md - Dump the raw memory of the current state
    !stackdump | !sd - Dump the raw stack of the current state
    !rawstack <var> | !rs <var> - Display the raw value of a variable's stack
allocation. For variables that are > 32 bytes in length, this will display
their memory pointer.
```

General

`!help | !h`

Display all commands.

`!quit | !q`

Quit Chisel.

`!exec <command> [args] | !e <command> [args]`

Execute a shell command and print the output.

Example:

```
➜ !e ls
CHANGELOG.md
LICENSE
README.md
TESTS.md
artifacts
cache
contracts
crytic-export
deploy
deploy-config
deployments
dist
echidna.yaml
forge-artifacts
foundry.toml
hardhat.config.ts
layout-lock.json
node_modules
package.json
scripts
slither.config.json
slither.db.json
src
tasks
test-case-generator
tsconfig.build.json
tsconfig.build.tsbuildinfo
tsconfig.json
```

Session

`!clear | !c`

Clear current session source.

Under the hood, each Chisel session has an underlying contract that is altered as you input statements. This command clears this contract and resets your session to the default state.

`!source | !so`

Display the source code of the current session.

As mentioned above, each Chisel session has an underlying contract. This command will display the source code of this contract.

`!save [id] | !s [id]`

Save the current session to cache.

Chisel allows for caching sessions, which can be very useful if you are testing more complex logic in Chisel or if you want to return to a session at a later time. All cached Chisel sessions are stored in `~/.foundry/cache/chisel`.

If an `id` argument is not supplied, Chisel will automatically assign a numerical ID to the session you are saving.

`!load <id> | !l <id>`

Load a previous session ID from cache.

This command will load a previously cached session from the cache. Along with the session's source, all environment settings will also be loaded. The `id` argument must correspond with an existing cached session in the `~/.foundry/cache/chisel` directory.

`!list | !ls`

List all cached sessions.

This command will display all cached chisel sessions within the `~/.foundry/cache/chisel` directory.

`!clearcache | !cc`

Clear the chisel cache of all stored sessions.

Deletes all cache files within the `~/.foundry/cache/chisel` directory. These sessions are unrecoverable, so use this command with care.

`!export | !ex`

Export the current session source to a script file.

If `chisel` was executed from the root directory of a foundry project, it is possible to export your current session to a foundry script in the `scripts` dir of your project.

`!fetch <addr> <name> | !fe <addr> <name>`

Fetch the interface of a verified contract on Etherscan.

This command will attempt to parse the interface of a verified contract @ `<addr>` from the Etherscan API. If successful, the interface will be inserted into the session source with the name `<name>`.

At the moment, only interfaces of verified contracts on Ethereum mainnet can be fetched. In the future, Chisel will support fetching interfaces from multiple Etherscan-supported chains.

`!edit`

Open the current session's `run()` function in an editor.

chisel will use the editor defined in the `$EDITOR` environment variable.

Environment

```
!fork <url> | !f <url>
```

Fork an RPC for the current session. Supply 0 arguments to return to a local network.

Attempts to fork the state of the provided RPC. If no URL is provided, returns to using a blank, local devnet state.

```
!traces | !t
```

Enable / disable traces for the current session.

When tracing is enabled, foundry-style call tracing and logs will be printed after each statement is inserted.

Debug

```
!memdump | !md
```

Dump the raw memory of the current state.

Attempts to dump the raw memory of the machine state after the last instruction of the REPL contract's `run` function has finished executing.

```
!stackdump | !sd
```

Dump the raw stack of the current state.

Attempts to dump the raw stack of the machine state after the last instruction of the REPL contract's `run` function has finished executing.

```
!rawstack <var> | !rs <var>
```

Display the raw value of a variable's stack allocation. For variables that are > 32 bytes in length, this will display their memory pointer.

This command is useful when you want to view the full raw stack allocation for a variable that is less than 32 bytes in length.

Example:

```
➜ address addr
➜ assembly {
    addr := not(0)
}
➜ addr
Type: address
└ Data: 0xffffffffffffffffffffffffffffffffffffffff
➜ !rs addr
Type: bytes32
└ Data: 0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff
➜
```

Config Reference

- Overview
- Project
- Solidity Compiler
- Testing
- In-line test configuration
- Formatter
- Documentation Generator
- Etherscan

Config Overview

Foundry's configuration system allows you to configure its tools.

Profiles

Configuration can be arbitrarily namespaced into profiles. The default profile is named `default`, and all other profiles inherit values from this profile. Profiles are defined in the `profile` map.

To add a profile named `local`, you would add:

```
[profile.local]
```

You can select the profile to use by setting the `FOUNDRY_PROFILE` environment variable.

Global configuration

You can create a `foundry.toml` file in `~/.foundry` folder to configure Foundry globally.

Environment variables

Configuration can be overridden with `FOUNDRY_` and `DAPP_` prefixed environment variables.

Exceptions are:

- `FOUNDRY_FFI`, `DAPP_FFI`, `DAPP_TEST_FFI`
- `FOUNDRY_PROFILE`
- `FOUNDRY_REMAPPINGS`, `DAPP_REMAPPINGS`
- `FOUNDRY_LIBRARIES`, `DAPP_LIBRARIES`
- `FOUNDRY_FS_PERMISSIONS`, `DAPP_FS_PERMISSIONS`, `DAPP_TEST_FS_PERMISSIONS`
- `DAPP_TEST_CACHE`
- `DAPP_TEST_FUZZ_RUNS`
- `DAPP_TEST_FUZZ_DEPTH`

Configuration format

Configuration files are written in the TOML format, with simple key-value pairs inside of sections.

This page describes each configuration key in detail. To see the default values, either refer to the specific key in this document, or see the default config.

Configuration keys

This section documents all configuration keys. All configuration keys must live under a profile, such as `default`.

Project

Configuration related to the project in general.

`src`

- Type: string
- Default: src
- Environment: `FOUNDRY_SRC` or `DAPP_SRC`

The path to the contract sources relative to the root of the project.

`test`

- Type: string
- Default: test
- Environment: `FOUNDRY_TEST` or `DAPP_TEST`

The path to the test contract sources relative to the root of the project.

`script`

- Type: string
- Default: script
- Environment: `FOUNDRY_SCRIPT` or `DAPP_SCRIPT`

The path to the script contract sources relative to the root of the project.

`out`

- Type: string
- Default: out
- Environment: `FOUNDRY_OUT` or `DAPP_OUT`

The path to put contract artifacts in, relative to the root of the project.

`libs`

- Type: array of strings (paths)
- Default: lib
- Environment: `FOUNDRY_LIBS` or `DAPP_LIBS`

An array of paths that contain libraries, relative to the root of the project.

`cache`

- Type: boolean
- Default: true
- Environment: `FOUNDRY_CACHE` or `DAPP_CACHE`

Whether or not to enable caching. If enabled, the result of compiling sources, tests, and dependencies, are cached in `cache`.

`cache_path`

- Type: string
- Default: cache
- Environment: `FOUNDRY_CACHE_PATH` or `DAPP_CACHE_PATH`

The path to the cache, relative to the root of the project.

`broadcast`

- Type: string
- Default: broadcast

The path to the broadcast transaction logs, relative to the root of the project.

`force`

- Type: boolean
- Default: false
- Environment: `FOUNDRY_FORCE` or `DAPP_FORCE`

Whether or not to perform a clean build, discarding the cache.

Solidity compiler

Configuration related to the behavior of the Solidity compiler.

Sections

- General
- Optimizer
- Model Checker

General

Configuration related to the behavior of the Solidity compiler.

`remappings`

- Type: array of strings (remappings)
- Default: none
- Environment: `FOUNDRY_REMAPPINGS` or `DAPP_REMAPPINGS`

An array of remappings in the following format: `<name>=<target>` .

A remapping *remaps* Solidity imports to different directories. For example, the following remapping

```
@openzeppelin/=node_modules/@openzeppelin/openzeppelin-contracts/
```

with an import like

```
import "@openzeppelin/contracts/utils/Context.sol";
```

becomes

```
import "node_modules/@openzeppelin/openzeppelin-
contracts/contracts/utils/Context.sol";
```

`auto_detect_remappings`

- Type: boolean
- Default: true
- Environment: `FOUNDRY_AUTO_DETECT_REMAPPINGS` or `DAPP_AUTO_DETECT_REMAPPINGS`

If enabled, Foundry will automatically try auto-detect remappings by scanning the `libs` folder(s).

If set to `false`, only the remappings in `foundry.toml` and `remappings.txt` are used.

`allow_paths`

- Type: array of strings (paths)
- Default: none
- Environment: `FOUNDRY_ALLOW_PATHS` or `DAPP_ALLOW_PATHS`

Tells solc to allow reading source files from additional directories. This is mainly relevant for complex workspaces managed by `pnpm` or similar.

See also solc allowed-paths

`include_paths`

- Type: array of strings (paths)
- Default: none
- Environment: `FOUNDRY_INCLUDE_PATHS` or `DAPP_INCLUDE_PATHS`

Make an additional source directory available to the default import callback. Use this option if you want to import contracts whose location is not fixed in relation to your main source tree, e.g. third-party libraries installed using a package manager. Can be used multiple times. Can only be used if base path has a non-empty value.

See also solc path resolution

`libraries`

- Type: array of strings (libraries)
- Default: none
- Environment: `FOUNDRY_LIBRARIES` or `DAPP_LIBRARIES`

An array of libraries to link against in the following format: `<file>:<lib>:<address>`, for example: `src/MyLibrary.sol:MyLibrary:0xfD88CeE74f7D78697775aBDAE53f9Da1559728E4`.

`solc_version`

- Type: string (semver)
- Default: none
- Environment: `FOUNDRY_SOLC_VERSION` or `DAPP_SOLC_VERSION`

If specified, overrides the auto-detection system (more below) and uses a single Solidity compiler version for the project.

Only strict versions are supported (i.e. `0.8.11` is valid, but `^0.8.0` is not).

`auto_detect_solc`

- Type: boolean
- Default: true
- Environment: `FOUNDRY_AUTO_DETECT_SOLC` or `DAPP_AUTO_DETECT_SOLC`

If enabled, Foundry will automatically try to resolve appropriate Solidity compiler versions to compile your project.

This key is ignored if `solc_version` is set.

`offline`

- Type: boolean
- Default: false
- Environment: `FOUNDRY_OFFLINE` or `DAPP_OFFLINE`

If enabled, Foundry will not attempt to download any missing solc versions.

If both `offline` and `auto-detect-solc` are set to `true`, the required version(s) of solc will be auto detected but any missing versions will *not* be installed.

`ignored_warnings_from`

- Type: array of strings (file paths)
- Default: none
- Environment: `FOUNDRY_IGNORED_WARNINGS_FROM` OR `DAPP_IGNORED_WARNINGS_FROM`

An array of file paths from which warnings should be ignored during the compulation process. This is useful when you have a specific directories of files that produce known warning and you wish to suppress these warnings without affecting others.

Each entry in the array should be a path to a directory or a specific file. For Example:

```
ignored_warnings_from = ["path/to/warnings/file1.sol",
"path/to/warnings/file2.sol"]
```

This configuration will cause the compiler to ignore any warnings that originate from the specified paths.

`ignored_error_codes`

- Type: array of integers/strings
- Default: none for source, SPDX license identifiers and contract size for tests
- Environment: `FOUNDRY_IGNORED_ERROR_CODES` or `DAPP_IGNORED_ERROR_CODES`

An array of Solidity compiler error codes to ignore during build, such as warnings.

Valid values are:

- `license` : 1878
- `code-size` : 5574
- `func-mutability` : 2018
- `unused-var` : 2072
- `unused-param` : 5667
- `unused-return` : 9302
- `virtual-interfaces` : 5815
- `missing-receive-ether` : 3628
- `shadowing` : 2519
- `same-varname` : 8760
- `unnamed-return` : 6321
- `unreachable` : 5740
- `pragma-solidity` : 3420

`deny_warnings`

- Type: boolean
- Default: false
- Environment: `FOUNDRY_DENY_WARNINGS` or `DAPP_DENY_WARNINGS`

If enabled, Foundry will treat Solidity compiler warnings as errors, stopping artifacts from being written to disk.

`evm_version`

- Type: string
- Default: paris
- Environment: `FOUNDRY_EVM_VERSION` or `DAPP_EVM_VERSION`

The EVM version to use during tests. The value **must** be an EVM hardfork name, such as `london`, `byzantium`, etc.

`revert_strings`

- Type: string
- Default: default
- Environment: `FOUNDRY_REVERT_STRINGS` or `DAPP_REVERT_STRINGS`

Possible values are:

- `default` does not inject compiler-generated revert strings and keeps user-supplied ones.

- `strip` removes all revert strings (if possible, i.e. if literals are used) keeping side-effects.
- `debug` injects strings for compiler-generated internal reverts, implemented for ABI encoders V1 and V2 for now.
- `verboseDebug` even appends further information to user-supplied revert strings (not yet implemented).

`extra_output_files`

- Type: array of strings
- Default: none
- Environment: N/A

Extra output from the Solidity compiler that should be written to files in the artifacts directory.

Valid values are:

- `metadata` : Written as a `metadata.json` file in the artifacts directory
- `ir` : Written as a `.ir` file in the artifacts directory
- `irOptimized` : Written as a `.iropt` file in the artifacts directory
- `ewasm` : Written as a `.ewasm` file in the artifacts directory
- `evm.assembly` : Written as a `.asm` file in the artifacts directory

`extra_output`

- Type: array of strings
- Default: see below
- Environment: N/A

Extra output to include in the contract's artifact.

The following values are always set, since they're required by Forge:

```
extra_output = [
  "abi",
  "evm.bytecode",
  "evm.deployedBytecode",
  "evm.methodIdentifiers",
]
```

For a list of valid values, see the [Solidity docs][output-desc].

`bytecode_hash`

- Type: string
- Default: ipfs

- Environment: `FOUNDRY_BYTECODE_HASH` or `DAPP_BYTECODE_HASH`

Determines the hash method for the metadata hash that is appended to the bytecode.

Valid values are:

- ipfs (default)
- bzzr1
- none

`sparse_mode`

- Type: boolean
- Default: false
- Environment: `FOUNDRY_SPARSE_MODE` or `DAPP_SPARSE_MODE`

Enables sparse mode for builds.

Optimizer

Configuration related to the Solidity optimizer.

`optimizer`

- Type: boolean
- Default: true
- Environment: `FOUNDRY_OPTIMIZER` or `DAPP_OPTIMIZER`

Whether or not to enable the Solidity optimizer.

`optimizer_runs`

- Type: integer
- Default: 200
- Environment: `FOUNDRY_OPTIMIZER_RUNS` or `DAPP_OPTIMIZER_RUNS`

The amount of optimizer runs to perform.

`via_ir`

- Type: boolean
- Default: false
- Environment: `FOUNDRY_VIA_IR` or `DAPP_VIA_IR`

If set to true, changes compilation pipeline to go through the new IR optimizer.

`[optimizer_details]`

The optimizer details section is used to tweak how the Solidity optimizer behaves. There are several configurable values in this section (each of them are booleans):

- `peephole`
- `inliner`
- `jumpdest_remover`
- `order_literals`
- `deduplicate`
- `cse`
- `constant_optimizer`
- `yul`

Refer to the Solidity compiler input description for the default values.

`[optimizer_details.yul_details]`

The Yul details subsection of the optimizer details section is used to tweak how the new IR optimizer behaves. There are two configuration values:

- `stack_allocation` : Tries to improve the allocation of stack slots by freeing them up earlier.
- `optimizer_steps` : Selects the optimizer steps to be applied.

Refer to the Solidity compiler input description for the default values.

ℹ️ **Note** If you encounter compiler errors when using `via_ir` , explicitly enable the legacy `optimizer` and leave `optimizer_steps` as an empty string

---

Model checker

The Solidity model checker is a built-in opt-in module that is available in Solidity compilers for OSX and Linux. Learn more about the model checker in the Solidity compiler documentation

---

ℹ️ **Note** The model checker requires `z3` version 4.8.8 or 4.8.14 on Linux.

The model checker settings are configured in the `[model_checker]` section of the configuration.

The model checker will run when `forge build` is invoked, and any findings will show up as warnings.

These are the recommended settings when using the model checker:

```
[profile.default.model_checker]
contracts = {'/path/to/project/src/Contract.sol' = ['Contract']}
engine = 'chc'
timeout = 10000
targets = ['assert']
```

Setting which contract should be verified is extremely important, otherwise all available contracts will be verified which may take a long time.

The recommended engine is `chc`, but `bmc` and `all` (which runs both) are also accepted.

It is also important to set a proper timeout (given in milliseconds), since the default time given to the underlying solver may not be enough.

If no verification targets are given, only assertions will be checked.

`[model_checker]`

The following keys are available in the model checker section.

`model_checker.contracts`

- Type: table
- Default: all
- Environment: N/A

Specifies what contracts the model checker will analyze.

The key of the table is the path to a source file, and the value is an array of contract names to check.

For example:

```
[profile.default.model_checker]
contracts = { "src/MyContracts.sol" = ["ContractA", "ContractB"] }
```

`model_checker.div_mod_with_slacks`

- Type: boolean
- Default: false
- Environment: N/A

Sets how division and modulo operations should be encoded.

Refer to the Solidity documentation for more information.

`model_checker.engine`

- Type: string (see below)
- Default: all
- Environment: N/A

Specifies the model checker engine to run. Valid values are:

- `chc` : The constrained horn clauses engine
- `bmc` : The bounded model checker engine
- `all` : Runs both engines

Refer to the Solidity documentation for more information on the engines.

`model_checker.invariants`

- Type: array of strings
- Default: N/A
- Environment: N/A

Sets the model checker invariants. Valid values are:

- `contract` : Contract Invariants
- `reentrancy` : Reentrancy Properties

Refer to the Solidity documentation for more information on the invariants.

`model_checker.show_unproved`

- Type: boolean
- Default: false
- Environment: N/A

Whether or not to output all unproved targets.

Refer to the Solidity documentation for more information.

`model_checker.solvers`

- Type: array of strings
- Default: N/A
- Environment: N/A

Sets the model checker solvers. Valid values are:

- `cvc4`
- `eld` : introduced in v0.8.18
- `smtlib2`

- `z3`

Refer to the Solidity documentation for more information.

`model_checker.timeout`

- Type: number (milliseconds)
- Default: N/A
- Environment: N/A

Sets the timeout for the underlying model checker engines (in milliseconds).

`model_checker.targets`

- Type: array of strings
- Default: assert
- Environment: N/A

Sets the model checker targets. Valid values are:

- `assert` : Assertions
- `underflow` : Arithmetic underflow
- `overflow` : Arithmetic overflow
- `divByZero` : Division by zero
- `constantCondition` : Trivial conditions and unreachable code
- `popEmptyArray` : Popping an empty array
- `outOfBounds` : Out of bounds array/fixed bytes index access
- `default` : All of the above (note: not the default for Forge)

Testing

Configuration related to the behavior of `forge test`.

Sections

- General
- Fuzz
- Invariant

General

`verbosity`

- Type: integer
- Default: 0
- Environment: `FOUNDRY_VERBOSITY` or `DAPP_VERBOSITY`

The verbosity level to use during tests.

- **Level 2 (`-vv`)**: Logs emitted during tests are also displayed.
- **Level 3 (`-vvv`)**: Stack traces for failing tests are also displayed.
- **Level 4 (`-vvvv`)**: Stack traces for all tests are displayed, and setup traces for failing tests are displayed.
- **Level 5 (`-vvvvv`)**: Stack traces and setup traces are always displayed.

`ffi`

- Type: boolean
- Default: false
- Environment: `FOUNDRY_FFI` or `DAPP_FFI`

Whether or not to enable the `ffi` cheatcode.

Warning: Enabling this cheatcode has security implications for your project, as it allows tests to execute arbitrary programs on your computer.

`sender`

- Type: string (address)
- Default: 0x1804c8AB1F12E6bbf3894d4083f33e07309d1f38
- Environment: `FOUNDRY_SENDER` or `DAPP_SENDER`

The value of `msg.sender` in tests.

`tx_origin`

- Type: string (address)
- Default: 0x1804c8AB1F12E6bbf3894d4083f33e07309d1f38
- Environment: `FOUNDRY_TX_ORIGIN` or `DAPP_TX_ORIGIN`

The value of `tx.origin` in tests.

`initial_balance`

- Type: string (hexadecimal)
- Default: 0xffffffffffffffffffffffff
- Environment: `FOUNDRY_INITIAL_BALANCE` or `DAPP_INITIAL_BALANCE`

The initial balance of the test contracts in wei, written in hexadecimal.

`block_number`

- Type: integer
- Default: 1
- Environment: `FOUNDRY_BLOCK_NUMBER` or `DAPP_BLOCK_NUMBER`

The value of `block.number` in tests.

`chain_id`

- Type: integer
- Default: 31337
- Environment: `FOUNDRY_CHAIN_ID` or `DAPP_CHAIN_ID`

The value of the `chainid` opcode in tests.

`gas_limit`

- Type: integer or string
- Default: 9223372036854775807
- Environment: `FOUNDRY_GAS_LIMIT` or `DAPP_GAS_LIMIT`

The gas limit for each test case.

> ℹ️ **Note**
>
> Due to a limitation in a dependency of Forge, you **cannot raise the gas limit** beyond the default without changing the value to a string.

In order to use higher gas limits use a string:

```
gas_limit = "18446744073709551615" # u64::MAX
```

`gas_price`

- Type: integer
- Default: 0
- Environment: `FOUNDRY_GAS_PRICE` or `DAPP_GAS_PRICE`

The price of gas (in wei) in tests.

`block_base_fee_per_gas`

- Type: integer
- Default: 0
- Environment: `FOUNDRY_BLOCK_BASE_FEE_PER_GAS` or `DAPP_BLOCK_BASE_FEE_PER_GAS`

The base fee per gas (in wei) in tests.

`block_coinbase`

- Type: string (address)
- Default: 0x0000000000000000000000000000000000000000
- Environment: `FOUNDRY_BLOCK_COINBASE` or `DAPP_BLOCK_COINBASE`

The value of `block.coinbase` in tests.

`block_timestamp`

- Type: integer
- Default: 1
- Environment: `FOUNDRY_BLOCK_TIMESTAMP` or `DAPP_BLOCK_TIMESTAMP`

The value of `block.timestamp` in tests.

`block_difficulty`

- Type: integer
- Default: 0
- Environment: `FOUNDRY_BLOCK_DIFFICULTY` or `DAPP_BLOCK_DIFFICULTY`

The value of `block.difficulty` in tests.

`gas_reports`

- Type: array of strings (contract names)
- Default: ["*"]
- Environment: `FOUNDRY_GAS_REPORTS` or `DAPP_GAS_REPORTS`

The contracts to print gas reports for.

`no_storage_caching`

- Type: boolean
- Default: false
- Environment: `FOUNDRY_NO_STORAGE_CACHING` or `DAPP_NO_STORAGE_CACHING`

If set to `true`, then block data from RPC endpoints in tests will not be cached. Otherwise, the data is cached to `$HOME/.foundry/cache/<chain id>/<block number>`.

`[rpc_storage_caching]`

The `[rpc_storage_caching]` block determines what RPC endpoints are cached.

`rpc_storage_caching.chains`

- Type: string or array of strings (chain names)
- Default: all
- Environment: N/A

Determines what chains are cached. By default, all chains are cached.

Valid values are:

- "all"
- A list of chain names, e.g. `["optimism", "mainnet"]`

`rpc_storage_caching.endpoints`

- Type: string or array of regex patterns (to match URLs)
- Default: remote
- Environment: N/A

Determines what RPC endpoints are cached. By default, only remote endpoints are cached.

Valid values are:

- all
- remote (default)
- A list of regex patterns, e.g. `["localhost"]`

`eth_rpc_url`

- Type: string
- Default: none
- Environment: `FOUNDRY_ETH_RPC_URL` or `DAPP_ETH_RPC_URL`

The url of the rpc server that should be used for any rpc calls.

`etherscan_api_key`

- Type: string
- Default: none
- Environment: `FOUNDRY_ETHERSCAN_API_KEY` or `DAPP_ETHERSCAN_API_KEY`

The etherscan API key for RPC calls.

`match-test`

- Type: regex
- Default: none
- Environment: `FOUNDRY_MATCH_TEST` or `DAPP_MATCH_TEST`

Only run test methods matching regex. Equivalent to `forge test --match-test <TEST_PATTERN>`

`no-match-test`

- Type: regex
- Default: none
- Environment: `FOUNDRY_NO_MATCH_TEST` or `DAPP_NO_MATCH_TEST`

Only run test methods not matching regex. Equivalent to `forge test --no-match-test <TEST_PATTERN_INVERSE>`

`match-contract`

- Type: regex
- Default: none
- Environment: `FOUNDRY_MATCH_CONTRACT` or `DAPP_MATCH_CONTRACT`

Only run test methods in contracts matching regex. Equivalent to `forge test --match-contract <CONTRACT_PATTERN>`

`no-match-contract`

- Type: regex
- Default: none

- Environment: `FOUNDRY_NO_MATCH_CONTRACT` or `DAPP_NO_MATCH_CONTRACT`

Only run test methods in contracts not matching regex. Equivalent to `forge test --no-match-contract <CONTRACT_PATTERN_INVERSE>`

`match-path`

- Type: regex
- Default: none
- Environment: `FOUNDRY_MATCH_PATH` or `DAPP_MATCH_PATH`

Only runs test methods on files matching the path. Equivalent to `forge test --match-path <PATH_PATTERN>`

`no-match-path`

- Type: regex
- Default: none
- Environment: `FOUNDRY_NO_MATCH_PATH` or `DAPP_NO_MATCH_PATH`

Only runs test methods on files not matching the path. Equivalent to `forge test --no-match-path <PATH_PATTERN_INVERSE>`

`block_gas_limit`

- Type: integer
- Default: none
- Environment: `FOUNDRY_BLOCK_GAS_LIMIT` or `DAPP_BLOCK_GAS_LIMIT`

The block.gaslimit value during EVM execution.

`memory_limit`

- Type: integer
- Default: 33554432
- Environment: `FOUNDRY_MEMORY_LIMIT` or `DAPP_MEMORY_LIMIT`

The memory limit of the EVM in bytes.

`names`

- Type: boolean
- Default: false
- Environment: `FOUNDRY_NAMES` or `DAPP_NAMES`

Print compiled contract names.

`sizes`

- Type: boolean
- Default: false
- Environment: `FOUNDRY_SIZES` or `DAPP_SIZES`

Print compiled contract sizes.

`rpc_endpoints`

- Type: table of RPC endpoints
- Default: none
- Environment: none

This section lives outside of profiles and defines a table of RPC endpoints, where the key specifies the RPC endpoints's name and the value is the RPC endpoint itself.

The value can either be a valid RPC endpoint or a reference to an environment variable (wrapped with in `${}`).

These RPC endpoints can be used in tests and Solidity scripts (see `vm.rpc`).

The following example defines an endpoint named `optimism` and an endpoint named `mainnet` that references an environment variable `RPC_MAINNET`:

```
[rpc_endpoints]
optimism = "https://optimism.alchemyapi.io/v2/..."
mainnet = "${RPC_MAINNET}"
```

`prompt_timeout`

- Type: integer
- Default: 120
- Environment: `FOUNDRY_PROMPT_TIMEOUT`

The number of seconds to wait before `vm.prompt` reverts with a timeout.

Fuzz

Configuration values for `[fuzz]` section.

`runs`

- Type: integer
- Default: 256
- Environment: `FOUNDRY_FUZZ_RUNS` or `DAPP_FUZZ_RUNS`

The amount of fuzz runs to perform for each fuzz test case. Higher values gives more confidence in results at the cost of testing speed.

`max_test_rejects`

- Type: integer
- Default: 65536
- Environment: `FOUNDRY_FUZZ_MAX_TEST_REJECTS`

The maximum number of combined inputs that may be rejected before the test as a whole aborts. “Global” filters apply to the whole test case. If the test case is rejected, the whole thing is regenerated.

`seed`

- Type: string (hexadecimal)
- Default: none
- Environment: `FOUNDRY_FUZZ_SEED`

Optional seed for the fuzzing RNG algorithm.

`dictionary_weight`

- Type: integer (between 0 and 100)
- Default: 40
- Environment: `FOUNDRY_FUZZ_DICTIONARY_WEIGHT`

The weight of the dictionary. A higher dictionary weight will bias the fuzz inputs towards “interesting” values, e.g. boundary values like `type(uint256).max` or contract addresses from your environment.

`include_storage`

- Type: boolean
- Default: true
- Environment: `FOUNDRY_FUZZ_INCLUDE_STORAGE`

The flag indicating whether to include values from storage.

`include_push_bytes`

- Type: boolean
- Default: true
- Environment: `FOUNDRY_FUZZ_INCLUDE_PUSH_BYTES`

The flag indicating whether to include push bytes values.

Invariant

Configuration values for `[invariant]` section.

ℹ️ Note

Configuration for `[invariant]` section has the fallback logic for common config entries (`runs` , `seed` , `dictionary_weight` etc).

- If the entries are not set in either section, then the defaults will be used.
- If the entries are set in the `[fuzz]` section, but are not set in the `[invariant]` section, these values will automatically be set to the values specified in the `[fuzz]` section.
- For any profile other than `default` :
  - If at least one entry is set in the `[invariant]` (same as `[profile.default.invariant]`) section, then the values from `[invariant]` section will be used, including defaults.
  - If no entry is set in the `[invariant]` section, but there are entries in the `[fuzz]` (same as `[profile.default.fuzz]`) section, then the values from the `[fuzz]` section will be used.
  - If it's none of the cases described above, then the defaults will be used.

`runs`

- Type: integer
- Default: 256
- Environment: `FOUNDRY_INVARIANT_RUNS`

The number of runs that must execute for each invariant test group. See also fuzz.runs

`depth`

- Type: integer
- Default: 15
- Environment: `FOUNDRY_INVARIANT_DEPTH`

The number of calls executed to attempt to break invariants in one run.

`fail_on_revert`

- Type: boolean
- Default: false
- Environment: `FOUNDRY_INVARIANT_FAIL_ON_REVERT`

Fails the invariant fuzzing if a revert occurs.

`call_override`

- Type: boolean
- Default: false
- Environment: `FOUNDRY_INVARIANT_CALL_OVERRIDE`

Overrides unsafe external calls when running invariant tests, useful for e.g. performing reentrancy checks.

`dictionary_weight`

- Type: integer (between 0 and 100)
- Default: 80
- Environment: `FOUNDRY_INVARIANT_DICTIONARY_WEIGHT`

The weight of the dictionary. See also fuzz.dictionary_weight

`include_storage`

- Type: boolean
- Default: true
- Environment: `FOUNDRY_FUZZ_INCLUDE_STORAGE`

The flag indicating whether to include values from storage. See also fuzz.include_storage

`include_push_bytes`

- Type: boolean
- Default: true
- Environment: `FOUNDRY_FUZZ_INCLUDE_PUSH_BYTES`

The flag indicating whether to include push bytes values. See also fuzz.include_push_bytes

In-line test configuration

Foundry users are enabled to specify overall test configurations, using a combination of ENV variables and config statements in the `foundry.toml`. Checkout the `Testing reference` for a detailed description.

Despite this may work in the general case, some tests may need finer control over their configuration. For such reason Forge provides a way to specify per-test configs for invariant and fuzz testing scenarios.

Users can in-line test config statements directly in Solidity comments. This would affect the behavior of the `forge test` command for a specific test instance, as illustrated in the example below.

```
contract MyTest is Test {
  /// forge-config: default.fuzz.runs = 100
  /// forge-config: ci.fuzz.runs = 500
  function test_SimpleFuzzTest(uint256 x) public {
    // --- snip ---
  }
}
```

What we are asking here is to run our fuzzer `100` and `500` times for the `default` and `ci` profiles respectively. The interesting fact is that this would override any fuzz `runs` setup existing at a global level. All other configs would be inherited from the global context, making this acting as a fallback for all possible configurations.

Block comments

In-line test configurations can also be expressed in block comments, as illustrated in the example.

```
contract MyTest is Test {
  /**
   * forge-config: default.fuzz.runs = 1024
   * forge-config: default.fuzz.max-test-rejects = 500
   */
  function test_SimpleFuzzTest(uint256 x) public {
    // --- snip ---
  }
}
```

In-line fuzz configs

Users can specify the configs described in the table. Each statement must have a prefix of the form `forge-config: ${PROFILE}.fuzz.`

Parameter	Type	Description
`runs`	integer	The amount of fuzz runs to perform for this specific test case. `Reference` .
`max-test-rejects`	integer	The maximum number of combined inputs that may be rejected before the test as a whole aborts. `Reference` .

Fuzz config example

```
contract MyFuzzTest is Test {
  /// forge-config: default.fuzz.runs = 100
  /// forge-config: default.fuzz.max-test-rejects = 2
  function test_InlineConfig(uint256 x) public {
    // --- snip ---
  }
}
```

In-line invariant configs

Users can specify the configs described in the table. Each statement must have a prefix of the form `forge-config: ${PROFILE}.invariant.`

Parameter	Type	Description
`runs`	integer	The amount of invariant runs to perform for this specific test case. `Reference` .
`depth`	integer	The number of calls executed to attempt to break invariant in one run. `Reference` .
`fail-on-revert`	boolean	Fails the invariant fuzzing if a revert occurs. `Reference` .
`call-override`	boolean	Overrides unsafe external calls when running invariant test. `Reference` .

Invariant config example

```
contract MyInvariantTest is Test {
  /// forge-config: default.invariant.runs = 100
  /// forge-config: default.invariant.depth = 2
  /// forge-config: default.invariant.fail-on-revert = false
  /// forge-config: default.invariant.call-override = true
  function invariant_InlineConfig() public {
    // --- snip ---
  }
}
```

Formatter

Configuration related to the behavior of the Forge formatter. Each of these keys live under the `[fmt]` section.

`single_line_statement_blocks`

This allows configuring the style of statement blocks independently of the line length.

The `single_line_statement_blocks` configuration option in Foundry controls whether statement blocks (code blocks enclosed in `{ }`) will be formatted to a single line or multiple lines.

It has three possible values:

- `"single"` - Statement blocks will be formatted to a single line if possible.
- `"multi"` - Statement blocks will always be formatted to multiple lines.
- `"preserve"` - Keep the existing single/multi line formatting of statement blocks.

For example, with `"single"`:

```
if (true) { return true; }
```

With `"multi"`:

```
if (true) {
  return true;
}
```

And with `"preserve"`, it will keep the original formatting of the code.

`multiline_func_header`

This allows configuring the formatting of long function headers.

- Type: string

- Default: `attributes_first`
- Environment: `FOUNDRY_FMT_MULTILINE_FUNC_HEADER` or `DAPP_FMT_MULTILINE_FUNC_HEADER`

The `multiline_func_header` configuration option in Foundry controls how function headers (the line containing the function name, parameters, return values etc.) are formatted when they exceed the max line length.

Style of multiline function header in case it doesn't fit in one line. Valid possible values:

- `"attributes_first"` - (default) Write function attributes multiline first
- `"params_first"` - Break the function header into multiple lines, with each parameter on its own line. The function name stays on the first line.
- `"all"` - If function parameters or attributes are multiline, multiline everything

Style of multiline function header in case it doesn't fit in one line. Valid values are:

For example, with `"params_first"`:

```
function myFunction(
    uint256 param1,
    uint256 param2,
    uint256 param3
) public returns (uint256) {
  // ...
}
```

And with `"all"`:

```
function myFunction(
    uint256 param1,
    uint256 param2,
    uint256 param3
)
    public
    returns (uint256)
{
  // ...
}
```

sort_imports

---

helps organize imports and makes it easier to find specific imports quickly.

The `sort_imports` configuration option in Foundry controls whether import statements are sorted alphabetically within their import groups.

From the code snippets provided:

- In `fmt.rs`, `sort_imports` is a configuration option that can be set to `true` or `false`.
- In `formatter.rs`, the `sort_imports()` method is called to sort import statements alphabetically if `sort_imports` is enabled.
- It finds import groups separated by blank lines in the source code.
- Then sorts the import statements within each group alphabetically.

So in summary, enabling `sort_imports` will reorder solidity import statements alphabetically within their section groups, while preserving the relative ordering of the groups.

`contract_new_lines`

Controls the addition of blank lines before and after contract definitions.

The `contract_new_lines` configuration option in Foundry controls whether empty lines are added before and after contract definitions.

When enabled (`contract_new_lines = true`):

- An empty line will be added before the start of a contract definition.
- An empty line will also be added after the end of a contract definition.

For example, with `contract_new_lines` enabled:

```
// Empty line before contract contract MyContract { // ... } // Empty line
after contract
```

This helps visually separate contract definitions and improve readability.

When disabled (`contract_new_lines = false`), empty lines will not be added around contracts.

So in summary, the `contract_new_lines` option controls the addition of blank lines before and after contract definitions.

override_spacing

The `override_spacing` configuration option in Foundry controls whether a space is printed between the `override` keyword and the parent contracts when overriding a function or modifier.

When `override_spacing` is enabled:

- A space will be added after the `override` keyword.

For example:

```
contract Child is Parent {
  function foo() override (Parent) public { }
}
```

When disabled, there will be no space between `override` and the parent contracts:

```
contract Child is Parent {
  function foo() override(Parent) public { }
}
```

Enables configuring the styling of override declarations.

- `override_spacing = true` - Print a space after `override`
- `override_spacing = false` - No space after `override`

line_length

- Type: number
- Default: 120
- Environment: `FOUNDRY_FMT_LINE_LENGTH` or `DAPP_FMT_LINE_LENGTH`

Maximum line length where formatter will try to wrap the line.

tab_width

- Type: number
- Default: 4
- Environment: `FOUNDRY_FMT_TAB_WIDTH` or `DAPP_FMT_TAB_WIDTH`

Number of spaces per indentation level.

bracket_spacing

- Type: bool
- Default: false

- Environment: `FOUNDRY_FMT_BRACKET_SPACING` or `DAPP_FMT_BRACKET_SPACING`

Whether or not to print spaces between brackets.

`int_types`

- Type: string
- Default: `long`
- Environment: `FOUNDRY_FMT_INT_TYPES` or `DAPP_FMT_INT_TYPES`

Style of uint/int256 types. Valid values are:

- `long` (default): Use the explicit `uint256` or `int256`
- `short` : Use the implicit `uint` or `int`
- `preserve` : Use the type defined in the source code

`quote_style`

- Type: string
- Default: `double`
- Environment: `FOUNDRY_FMT_QUOTE_STYLE` or `DAPP_FMT_QUOTE_STYLE`

Defines the quotation mark style. Valid values are:

- `double` (default): Use double quotes where possible (`"`)
- `single` : Use single quotes where possible (`'`)
- `preserve` : Use quotation mark defined in the source code

`number_underscore`

- Type: string
- Default: `preserve`
- Environment: `FOUNDRY_FMT_NUMBER_UNDERSCORE` or `DAPP_FMT_NUMBER_UNDERSCORE`

Style of underscores in number literals. Valid values are:

- `preserve` (default): Use the underscores defined in the source code
- `thousands` : Add an underscore every thousand, if greater than 9999. i.e. `1000` is formatted as `1000` and `10000` as `10_000`
- `remove` : Remove all underscores

`wrap_comments`

- Type: bool
- Default: false
- Environment: `FOUNDRY_FMT_WRAP_COMMENTS` or `DAPP_FMT_WRAP_COMMENTS`

Whether or not to wrap comments on `line_length` reached.

`ignore`

- Type: array of strings (patterns)
- Default: `[]`
- Environment: `FOUNDRY_FMT_IGNORE` or `DAPP_FMT_IGNORE`

List of files to ignore when formatting. This is a comma separated list of glob patterns.

Documentation Generator

Configuration related to the behavior of the Forge documentation generator. These keys are set in `[doc]` section.

`out`

- Type: string
- Default: `docs`
- Environment: `FOUNDRY_DOC_OUT`

An output path for generated documentation.

`title`

- Type: string
- Environment: `FOUNDRY_DOC_TITLE`

Title for the generated documentation.

`book`

- Type: string
- Default: `./book.toml`
- Environment: `FOUNDRY_DOC_BOOK`

Path to user provided `book.toml`. It will be merged with default settings during doc generation.

`repository`

- Type: string
- Environment: `FOUNDRY_DOC_REPOSITORY`

The git repository URL. Used to provide links to git source files. If missing, `forge` will attempt to look up the current origin url and use its value.

`ignore`

- Type: array of strings (patterns)
- Default: `[]`
- Environment: `FOUNDRY_DOC_IGNORE`

List of files to ignore when generating documentation. This is a comma separated list of glob patterns.

Etherscan

Configuration related to Etherscan, such as API keys. This configuration is used in various places by Forge.

The `[etherscan]` section is a mapping of keys to Etherscan configuration tables. The Etherscan configuration tables hold the following keys:

- `key` (string) (**required**): The Etherscan API key for the given network. The value of this property can also point to an environment variable.
- `chain` : The chain name or ID of the chain this Etherscan configuration is for.
- `url` : The Etherscan API URL.

If the key of the configuration is a chain name, then `chain` is not required, otherwise it is. `url` can be used to explicitly set the Etherscan API URL for chains not natively supported by name.

Using TOML inline table syntax, all of these are valid:

```
[etherscan]
mainnet = { key = "${ETHERSCAN_MAINNET_KEY}" }
mainnet2 = { key = "ABCDEFG", chain = "mainnet" }
optimism = { key = "1234567" }
unknown_chain = { key = "ABCDEFG", url = "<etherscan api url for this chain>" }
```

Cheatcodes Reference

Cheatcodes give you powerful assertions, the ability to alter the state of the EVM, mock data, and more.

Cheatcodes are made available through use of the cheatcode address (`0x7109709ECfa91a80626fF3989D68f67F5b1DD12D`).

Note

If you encounter errors for this address when using fuzzed addresses in your tests, you may wish to exclude it from your fuzz tests by using the following line:

```
vm.assume(address_ != 0x7109709ECfa91a80626fF3989D68f67F5b1DD12D);
```

You can also access cheatcodes easily via `vm` available in Forge Standard Library's `Test` contract.

Forge Standard Library Cheatcodes

Forge Std implements wrappers around cheatcodes, which combine multiple standard cheatcodes to improve development experience. These are not technically cheatcodes, but rather compositions of Forge's cheatcodes.

You can view the list of Forge Standard Library's cheatcode wrappers in the references section. You can reference the Forge Std source code to learn more about how the wrappers work under the hood.

Cheatcode Types

Below are some subsections for the different Forge cheatcodes.

- Environment: Cheatcodes that alter the state of the EVM.
- Assertions: Cheatcodes that are powerful assertions
- Fuzzer: Cheatcodes that configure the fuzzer
- External: Cheatcodes that interact with external state (files, commands, ...)
- Utilities: Smaller utility cheatcodes
- Forking: Forking mode cheatcodes
- Snapshots: Snapshot cheatcodes
- RPC: RPC related cheatcodes

- File: Cheatcodes for working with files

Add a new cheatcode

If you need a new feature, consider contributing to the Foundry's codebase to add the cheatcode.

Cheatcodes Interface

This is a Solidity interface for all of the cheatcodes present in Forge.

```
interface CheatCodes {
    // This allows us to getRecordedLogs()
    struct Log {
        bytes32[] topics;
        bytes data;
    }

    // Possible caller modes for readCallers()
    enum CallerMode {
        None,
        Broadcast,
        RecurrentBroadcast,
        Prank,
        RecurrentPrank
    }

    enum AccountAccessKind {
        Call,
        DelegateCall,
        CallCode,
        StaticCall,
        Create,
        SelfDestruct,
        Resume
    }

    struct Wallet {
        address addr;
        uint256 publicKeyX;
        uint256 publicKeyY;
        uint256 privateKey;
    }

    struct ChainInfo {
        uint256 forkId;
        uint256 chainId;
    }

    struct AccountAccess {
        ChainInfo chainInfo;
        AccountAccessKind kind;
        address account;
        address accessor;
        bool initialized;
        uint256 oldBalance;
        uint256 newBalance;
        bytes deployedCode;
        uint256 value;
        bytes data;
        bool reverted;
        StorageAccess[] storageAccesses;
    }

    struct StorageAccess {
        address account;
        bytes32 slot;
        bool isWrite;
```

```
        bytes32 previousValue;
        bytes32 newValue;
        bool reverted;
    }

    // Derives a private key from the name, labels the account with that name,
and returns the wallet
    function createWallet(string calldata) external returns (Wallet memory);

    // Generates a wallet from the private key and returns the wallet
    function createWallet(uint256) external returns (Wallet memory);

    // Generates a wallet from the private key, labels the account with that
name, and returns the wallet
    function createWallet(uint256, string calldata) external returns (Wallet
memory);

    // Signs data, (Wallet, digest) => (v, r, s)
    function sign(Wallet calldata, bytes32) external returns (uint8, bytes32,
bytes32);

    // Get nonce for a Wallet
    function getNonce(Wallet calldata) external returns (uint64);

    // Set block.timestamp
    function warp(uint256) external;

    // Set block.number
    function roll(uint256) external;

    // Set block.basefee
    function fee(uint256) external;

    // Set block.difficulty
    // Does not work from the Paris hard fork and onwards, and will revert
instead.
    function difficulty(uint256) external;

    // Set block.prevrandao
    // Does not work before the Paris hard fork, and will revert instead.
    function prevrandao(bytes32) external;

    // Set block.chainid
    function chainId(uint256) external;

    // Loads a storage slot from an address
    function load(address account, bytes32 slot) external returns (bytes32);

    // Stores a value to an address' storage slot
    function store(address account, bytes32 slot, bytes32 value) external;

    // Signs data
    function sign(uint256 privateKey, bytes32 digest)
        external
        returns (uint8 v, bytes32 r, bytes32 s);

    // Computes address for a given private key
    function addr(uint256 privateKey) external returns (address);
```

```
    // Derive a private key from a provided mnemonic string,
    // or mnemonic file path, at the derivation path m/44'/60'/0'/0/{index}.
    function deriveKey(string calldata, uint32) external returns (uint256);
    // Derive a private key from a provided mnemonic string, or mnemonic file
path,
    // at the derivation path {path}{index}
    function deriveKey(string calldata, string calldata, uint32) external
returns (uint256);

    // Gets the nonce of an account
    function getNonce(address account) external returns (uint64);

    // Sets the nonce of an account
    // The new nonce must be higher than the current nonce of the account
    function setNonce(address account, uint64 nonce) external;

    // Performs a foreign function call via terminal
    function ffi(string[] calldata) external returns (bytes memory);

    // Set environment variables, (name, value)
    function setEnv(string calldata, string calldata) external;

    // Read environment variables, (name) => (value)
    function envBool(string calldata) external returns (bool);
    function envUint(string calldata) external returns (uint256);
    function envInt(string calldata) external returns (int256);
    function envAddress(string calldata) external returns (address);
    function envBytes32(string calldata) external returns (bytes32);
    function envString(string calldata) external returns (string memory);
    function envBytes(string calldata) external returns (bytes memory);

    // Read environment variables as arrays, (name, delim) => (value[])
    function envBool(string calldata, string calldata)
        external
        returns (bool[] memory);
    function envUint(string calldata, string calldata)
        external
        returns (uint256[] memory);
    function envInt(string calldata, string calldata)
        external
        returns (int256[] memory);
    function envAddress(string calldata, string calldata)
        external
        returns (address[] memory);
    function envBytes32(string calldata, string calldata)
        external
        returns (bytes32[] memory);
    function envString(string calldata, string calldata)
        external
        returns (string[] memory);
    function envBytes(string calldata, string calldata)
        external
        returns (bytes[] memory);

    // Read environment variables with default value, (name, value) => (value)
    function envOr(string calldata, bool) external returns (bool);
    function envOr(string calldata, uint256) external returns (uint256);
```

```
    function envOr(string calldata, int256) external returns (int256);
    function envOr(string calldata, address) external returns (address);
    function envOr(string calldata, bytes32) external returns (bytes32);
    function envOr(string calldata, string calldata) external returns (string
memory);
    function envOr(string calldata, bytes calldata) external returns (bytes
memory);

    // Read environment variables as arrays with default value, (name, value[])
=> (value[])
    function envOr(string calldata, string calldata, bool[] calldata) external
returns (bool[] memory);
    function envOr(string calldata, string calldata, uint256[] calldata)
external returns (uint256[] memory);
    function envOr(string calldata, string calldata, int256[] calldata)
external returns (int256[] memory);
    function envOr(string calldata, string calldata, address[] calldata)
external returns (address[] memory);
    function envOr(string calldata, string calldata, bytes32[] calldata)
external returns (bytes32[] memory);
    function envOr(string calldata, string calldata, string[] calldata)
external returns (string[] memory);
    function envOr(string calldata, string calldata, bytes[] calldata) external
returns (bytes[] memory);

    // Convert Solidity types to strings
    function toString(address) external returns(string memory);
    function toString(bytes calldata) external returns(string memory);
    function toString(bytes32) external returns(string memory);
    function toString(bool) external returns(string memory);
    function toString(uint256) external returns(string memory);
    function toString(int256) external returns(string memory);

    // Sets the *next* call's msg.sender to be the input address
    function prank(address) external;

    // Sets all subsequent calls' msg.sender to be the input address
    // until `stopPrank` is called
    function startPrank(address) external;

    // Sets the *next* call's msg.sender to be the input address,
    // and the tx.origin to be the second input
    function prank(address, address) external;

    // Sets all subsequent calls' msg.sender to be the input address until
    // `stopPrank` is called, and the tx.origin to be the second input
    function startPrank(address, address) external;

    // Resets subsequent calls' msg.sender to be `address(this)`
    function stopPrank() external;

    // Reads the current `msg.sender` and `tx.origin` from state and reports if
there is any active caller modification
    function readCallers() external returns (CallerMode callerMode, address
msgSender, address txOrigin);

    // Sets an address' balance
    function deal(address who, uint256 newBalance) external;
```

```
    // Sets an address' code
    function etch(address who, bytes calldata code) external;

    // Marks a test as skipped. Must be called at the top of the test.
    function skip(bool skip) external;

    // Expects an error on next call
    function expectRevert() external;
    function expectRevert(bytes calldata) external;
    function expectRevert(bytes4) external;

    // Record all storage reads and writes
    function record() external;

    // Gets all accessed reads and write slot from a recording session,
    // for a given address
    function accesses(address)
        external
        returns (bytes32[] memory reads, bytes32[] memory writes);

    // Record all account accesses as part of CREATE, CALL or SELFDESTRUCT
opcodes in order,
    // along with the context of the calls.
    function startStateDiffRecording() external;

    // Returns an ordered array of all account accesses from a
`startStateDiffRecording` session.
    function stopAndReturnStateDiff() external returns (AccountAccess[] memory
accesses);

    // Record all the transaction logs
    function recordLogs() external;

    // Gets all the recorded logs
    function getRecordedLogs() external returns (Log[] memory);

    // Prepare an expected log with the signature:
    //   (bool checkTopic1, bool checkTopic2, bool checkTopic3, bool
checkData).
    //
    // Call this function, then emit an event, then call a function.
    // Internally after the call, we check if logs were emitted in the expected
order
    // with the expected topics and data (as specified by the booleans)
    //
    // The second form also checks supplied address against emitting contract.
    function expectEmit(bool, bool, bool, bool) external;
    function expectEmit(bool, bool, bool, bool, address) external;

    // Mocks a call to an address, returning specified data.
    //
    // Calldata can either be strict or a partial match, e.g. if you only
    // pass a Solidity selector to the expected calldata, then the entire
Solidity
    // function will be mocked.
    function mockCall(address, bytes calldata, bytes calldata) external;
```

```
    // Reverts a call to an address, returning the specified error
    //
    // Calldata can either be strict or a partial match, e.g. if you only
    // pass a Solidity selector to the expected calldata, then the entire
Solidity
    // function will be mocked.
    function mockCallRevert(address where, bytes calldata data, bytes calldata
retdata) external;

    // Clears all mocked and reverted mocked calls
    function clearMockedCalls() external;

    // Expect a call to an address with the specified calldata.
    // Calldata can either be strict or a partial match
    function expectCall(address callee, bytes calldata data) external;
    // Expect a call to an address with the specified
    // calldata and message value.
    // Calldata can either be strict or a partial match
    function expectCall(address callee, uint256, bytes calldata data) external;

    // Gets the _creation_ bytecode from an artifact file. Takes in the
relative path to the json file
    function getCode(string calldata) external returns (bytes memory);
    // Gets the _deployed_ bytecode from an artifact file. Takes in the
relative path to the json file
    function getDeployedCode(string calldata) external returns (bytes memory);

    // Label an address in test traces
    function label(address addr, string calldata label) external;

    // Retrieve the label of an address
    function getLabel(address addr) external returns (string memory);

    // When fuzzing, generate new inputs if conditional not met
    function assume(bool) external;

    // Set block.coinbase (who)
    function coinbase(address) external;

    // Using the address that calls the test contract or the address provided
    // as the sender, has the next call (at this call depth only) create a
    // transaction that can later be signed and sent onchain
    function broadcast() external;
    function broadcast(address) external;

    // Using the address that calls the test contract or the address provided
    // as the sender, has all subsequent calls (at this call depth only) create
    // transactions that can later be signed and sent onchain
    function startBroadcast() external;
    function startBroadcast(address) external;
    function startBroadcast(uint256 privateKey) external;

    // Stops collecting onchain transactions
    function stopBroadcast() external;

    // Reads the entire content of file to string, (path) => (data)
    function readFile(string calldata) external returns (string memory);
    // Get the path of the current project root
```

```
    function projectRoot() external returns (string memory);
    // Reads next line of file to string, (path) => (line)
    function readLine(string calldata) external returns (string memory);
    // Writes data to file, creating a file if it does not exist, and entirely
replacing its contents if it does.
    // (path, data) => ()
    function writeFile(string calldata, string calldata) external;
    // Writes line to file, creating a file if it does not exist.
    // (path, data) => ()
    function writeLine(string calldata, string calldata) external;
    // Closes file for reading, resetting the offset and allowing to read it
from beginning with readLine.
    // (path) => ()
    function closeFile(string calldata) external;
    // Removes file. This cheatcode will revert in the following situations,
but is not limited to just these cases:
    // - Path points to a directory.
    // - The file doesn't exist.
    // - The user lacks permissions to remove the file.
    // (path) => ()
    function removeFile(string calldata) external;
    // Returns true if the given path points to an existing entity, else
returns false
    // (path) => (bool)
    function exists(string calldata) external returns (bool);
    // Returns true if the path exists on disk and is pointing at a regular
file, else returns false
    // (path) => (bool)
    function isFile(string calldata) external returns (bool);
    // Returns true if the path exists on disk and is pointing at a directory,
else returns false
    // (path) => (bool)
    function isDir(string calldata) external returns (bool);

    // Return the value(s) that correspond to 'key'
    function parseJson(string memory json, string memory key) external returns
(bytes memory);
    // Return the entire json file
    function parseJson(string memory json) external returns (bytes memory);
    // Check if a key exists in a json string
    function keyExists(string memory json, string memory key) external returns
(bytes memory);
    // Get list of keys in a json string
    function parseJsonKeys(string memory json, string memory key) external
returns (string[] memory);

    // Snapshot the current state of the evm.
    // Returns the id of the snapshot that was created.
    // To revert a snapshot use `revertTo`
    function snapshot() external returns (uint256);
    // Revert the state of the evm to a previous snapshot
    // Takes the snapshot id to revert to.
    // This deletes the snapshot and all snapshots taken after the given
snapshot id.
    function revertTo(uint256) external returns (bool);

    // Creates a new fork with the given endpoint and block,
    // and returns the identifier of the fork
```

```
    function createFork(string calldata, uint256) external returns (uint256);
    // Creates a new fork with the given endpoint and the _latest_ block,
    // and returns the identifier of the fork
    function createFork(string calldata) external returns (uint256);

    // Creates _and_ also selects a new fork with the given endpoint and block,
    // and returns the identifier of the fork
    function createSelectFork(string calldata, uint256)
        external
        returns (uint256);
    // Creates _and_ also selects a new fork with the given endpoint and the
    // latest block and returns the identifier of the fork
    function createSelectFork(string calldata) external returns (uint256);

    // Takes a fork identifier created by `createFork` and
    // sets the corresponding forked state as active.
    function selectFork(uint256) external;

    // Returns the currently active fork
    // Reverts if no fork is currently active
    function activeFork() external returns (uint256);

    // Updates the currently active fork to given block number
    // This is similar to `roll` but for the currently active fork
    function rollFork(uint256) external;
    // Updates the given fork to given block number
    function rollFork(uint256 forkId, uint256 blockNumber) external;

    // Fetches the given transaction from the active fork and executes it on
the current state
    function transact(bytes32) external;
    // Fetches the given transaction from the given fork and executes it on the
current state
    function transact(uint256, bytes32) external;

    // Marks that the account(s) should use persistent storage across
    // fork swaps in a multifork setup, meaning, changes made to the state
    // of this account will be kept when switching forks
    function makePersistent(address) external;
    function makePersistent(address, address) external;
    function makePersistent(address, address, address) external;
    function makePersistent(address[] calldata) external;
    // Revokes persistent status from the address, previously added via
`makePersistent`
    function revokePersistent(address) external;
    function revokePersistent(address[] calldata) external;
    // Returns true if the account is marked as persistent
    function isPersistent(address) external returns (bool);

    /// Returns the RPC url for the given alias
    function rpcUrl(string calldata) external returns (string memory);
    /// Returns all rpc urls and their aliases `[alias, url][]`
    function rpcUrls() external returns (string[2][] memory);
}
```

Environment

- `warp`
- `roll`
- `fee`
- `difficulty`
- `prevrandao`
- `chainId`
- `store`
- `load`
- `etch`
- `deal`
- `prank`
- `startPrank`
- `stopPrank`
- `readCallers`
- `record`
- `accesses`
- `recordLogs`
- `getRecordedLogs`
- `setNonce`
- `getNonce`
- `mockCall`
- `mockCallRevert`
- `clearMockedCalls`
- `coinbase`
- `broadcast`
- `startBroadcast`
- `stopBroadcast`
- `pauseGasMetering`
- `resumeGasMetering`
- `txGasPrice`
- `startStateDiffRecording`
- `stopAndReturnStateDiff`

warp

Signature

```
function warp(uint256) external;
```

Description

Sets `block.timestamp`.

Examples

```
vm.warp(1641070800);
emit log_uint(block.timestamp); // 1641070800
```

SEE ALSO

Forge Standard Library

`skip`, `rewind`

roll

Signature

```
function roll(uint256) external;
```

Description

Sets `block.number`.

Examples

```
vm.roll(100);
emit log_uint(block.number); // 100
```

SEE ALSO

- rollFork

fee

Signature

```
function fee(uint256) external;
```

Description

Sets `block.basefee`.

Examples

```
vm.fee(25 gwei);
emit log_uint(block.basefee); // 25000000000
```

getBlockTimestamp

Signature

```
function getBlockTimestamp() external view returns (uint256 timestamp);
```

Description

Gets the current `block.timestamp`. This is useful in cases where `vm.warp` along with `--via-ir` compilation is used, as `block.timestamp` is assumed to be a constant during a transaction. This means that on every forge test, multiple calls to `block.timestamp` would get optimized to just returning a constant value, instead of actually accessing the current `block.timestamp`. `vm.getBlockTimestamp()` avoids this optimization and returns the current `block.timestamp`.

Examples

```
assertEq(vm.getBlockTimestamp(), 1, "timestamp should be 1");
vm.warp(10);
assertEq(vm.getBlockTimestamp(), 10, "warp failed");
```

getBlockNumber

Signature

```
function getBlockNumber() external view returns (uint256 timestamp);
```

Description

Gets the current `block.number`. This is useful in cases where `vm.roll` along with `--via-ir` compilation is used, as `block.number` is assumed to be a constant during a transaction. This means that on every forge test, multiple calls to `block.number` would get optimized to just returning a constant value, instead of actually accessing the current `block.number`. `vm.getBlockNumber()` avoids this optimization and returns the current `block.number`.

Examples

```
uint256 height = vm.getBlockNumber();
assertEq(height, uint256(block.number));
vm.roll(10);
assertEq(vm.getBlockNumber(), 10);
```

`difficulty`

Signature

```
function difficulty(uint256) external;
```

Description

Sets `block.difficulty`.

If used with a post-merge EVM version (Paris and onwards), it will revert. In that case, use `vm.prevrandao` instead.

Examples

```
vm.difficulty(25);
emit log_uint(block.difficulty); // 25
```

prevrandao

Signature

```
function prevrandao(bytes32) external;
```

Description

Sets `block.prevrandao`.

If used with an EVM version previous to the Paris hard fork, it will revert. In that case, use `vm.difficulty` instead.

Examples

```
vm.prevrandao(bytes32(uint256(42)));
emit log_uint(block.prevrandao); // 42
```

chainId

Signature

```
function chainId(uint256) external;
```

Description

Sets `block.chainid`.

Examples

```
vm.chainId(31337);
emit log_uint(block.chainid); // 31337
```

store

Signature

```
function store(address account, bytes32 slot, bytes32 value) external;
```

Description

Stores the value `value` in storage slot `slot` on account `account`.

Examples

```
/// contract LeetContract {
///     uint256 private leet = 1337; // slot 0
/// }

vm.store(address(leetContract), bytes32(uint256(0)), bytes32(uint256(31337)));
bytes32 leet = vm.load(address(leetContract), bytes32(uint256(0)));
emit log_uint(uint256(leet)); // 31337
```

SEE ALSO

Forge Standard Library

Std Storage

load

Signature

```
function load(address account, bytes32 slot) external returns (bytes32);
```

Description

Loads the value from storage slot `slot` on account `account`.

Examples

```
/// contract LeetContract {
///     uint256 private leet = 1337; // slot 0
/// }

bytes32 leet = vm.load(address(leetContract), bytes32(uint256(0)));
emit log_uint(uint256(leet)); // 1337
```

SEE ALSO

Forge Standard Library

Std Storage

etch

Signature

```
function etch(address who, bytes calldata code) external;
```

Description

Sets the bytecode of an address `who` to `code`.

Examples

```
bytes memory code = address(awesomeContract).code;
address targetAddr = makeAddr("target");
vm.etch(targetAddr, code);
log_bytes(address(targetAddr).code); // 0x6080604052348015610010...
```

Using `vm.etch` for enabling custom precompiles

Some chains, like Blast or Arbitrum, run with custom precompiles. Foundry is operating on vanilla EVM and is not aware of those. If you are encountering reverts due to not available precompile, you can use `vm.etch` cheatcode to inject mock of the missing precompile to the address it is expected to appear at.

```
pragma solidity 0.8.10;

import "forge-std/Test.sol";

// Firstly, we implement a mock emulating the actual precompile behavior
contract YieldMock {
    address private constant blastContract =
0x4300000000000000000000000000000000000002;

    mapping(address => uint8) public getConfiguration;

    function configure(address contractAddress, uint8 flags) external returns
(uint256) {
        require(msg.sender == blastContract);

        getConfiguration[contractAddress] = flags;
        return 0;
    }

    function claim(address, address, uint256) external pure returns (uint256) {
        return 0;
    }

    function getClaimableAmount(address) external pure returns (uint256) {
        return 0;
    }
}

contract SomeBlastTest is Test {
    function setUp() public {
        vm.createSelectFork("blastRpcUrl");
        // Deploy mock of the precompile
        YieldMock yieldMock = new YieldMock();
        // Set mock bytecode to the expected precompile address
        vm.etch(0x0000000000000000000000000000000000000100,
address(yieldMock).code);
    }

    function testSomething() public {
        // Now we can interact with Blast contracts without reverts
    }
}
```

> Injecting mocks of precompiles might be tricky as such mocks will not fully emulate the actual precompile behavior on-chain.
>
> Mock in the case above will not cause the actual yield to be accrued if any yield mode is configured.

SEE ALSO

Forge Standard Library

- `deployCode`
- `deployCodeTo`

deal

Signature

```
function deal(address who, uint256 newBalance) external;
```

Description

Sets the balance of an address `who` to `newBalance`.

If the alternative signature of `deal` is used (defined in `StdCheats.sol`), then we can additionally specify ERC20 token address, as well as an option to update `totalSupply`.

Examples

```
address alice = makeAddr("alice");
emit log_address(alice);
vm.deal(alice, 1 ether);
log_uint256(alice.balance); // 1000000000000000000
```

```
address alice = makeAddr("alice");
emit log_address(alice);
deal(address(DAI), alice, 1 ether); // import StdUtils.sol first
log_uint256(address(DAI).balanceOf(alice)); // 1000000000000000000
```

SEE ALSO

Forge Standard Library

`deal`, `hoax`, `startHoax`

prank

Signature

```
function prank(address) external;
```

```
function prank(address sender, address origin) external;
```

Description

Sets `msg.sender` to the specified address **for the next call**. "The next call" includes static calls as well, but not calls to the cheat code address.

If the alternative signature of `prank` is used, then `tx.origin` is set as well for the next call.

Examples

```
/// function withdraw() public {
///     require(msg.sender == owner);

vm.prank(owner);
myContract.withdraw(); // [PASS]
```

SEE ALSO

Forge Standard Library

`hoax`

startPrank

Signature

```
function startPrank(address) external;
```

```
function startPrank(address sender, address origin) external;
```

Description

Sets `msg.sender` **for all subsequent calls** until `stopPrank` is called.

If the alternative signature of `startPrank` is used, then `tx.origin` is set as well for all subsequent calls.

SEE ALSO

Forge Standard Library

`startHoax`, `changePrank`

stopPrank

Signature

```
function stopPrank() external;
```

Description

Stops an active prank started by `startPrank`, resetting `msg.sender` and `tx.origin` to the values before `startPrank` was called.

readCallers

Signature

```
enum CallerMode {
    None,
    Broadcast,
    RecurrentBroadcast,
    Prank,
    RecurrentPrank
}

function readCallers()
external
returns (CallerMode callerMode, address msgSender, address txOrigin);
```

Description

Reads the current `CallerMode`, `msg.sender`, and `tx.origin`.

The `CallerMode` enum indicates if there is an active caller modification and the type.

- If there is an active prank:
  - `callerMode` will be equal to:
    - `CallerMode.Prank` if the prank has been set with `prank`.
    - `CallerMode.RecurrentPrank` if the prank has been set with `startPrank`.
- If there is an active broadcast:
  - `callerMode` will be equal to:
    - `CallerMode.Broadcast` if the broadcast has been set with `broadcast`.
    - `CallerMode.RecurrentBroadcast` if the broadcast has been set with `startBroadcast`.
- If no caller modification is active:
  - `callerMode` will be equal to `CallerMode.None`.

Examples

```
CallerMode callerMode;
address msgSender;
address txOrigin;

// Example 1
(callerMode, msgSender, txOrigin) = vm.readCallers();
assertEq(callerMode, CallerMode.None);
assertEq(msgSender, defaultSenderAddress);
assertEq(txOrigin, defaultOriginAddress);

// Example 2
vm.prank(senderPrankAddress);
(callerMode, msgSender, txOrigin) = vm.readCallers();
assertEq(callerMode, CallerMode.Prank);
assertEq(msgSender, senderPrankAddress);
assertEq(txOrigin, defaultOriginAddress);

// Example 3
vm.prank(senderPrankAddress, originPrankAddress);
(callerMode, msgSender, txOrigin) = vm.readCallers();
assertEq(callerMode, CallerMode.Prank);
assertEq(msgSender, senderPrankAddress);
assertEq(txOrigin, originPrankAddress);

// Example 4
vm.broadcast(broadcastAddress);
(callerMode, msgSender, txOrigin) = vm.readCallers();
assertEq(callerMode, CallerMode.Broadcast);
assertEq(msgSender, broadcastAddress);
assertEq(txOrigin, broadcastAddress);
```

SEE ALSO

- prank
- startPrank
- broadcast
- startBroadcast

record

Signature

```
function record() external;
```

Description

Tell the VM to start recording all storage reads and writes. To access the reads and writes, use `accesses`.

ℹ️ **Note**

Every write also counts as an additional read.

Examples

```
/// contract NumsContract {
///     uint256 public num1 = 100; // slot 0
///     uint256 public num2 = 200; // slot 1
/// }

vm.record();
numsContract.num2();
(bytes32[] memory reads, bytes32[] memory writes) = vm.accesses(
  address(numsContract)
);
emit log_uint(uint256(reads[0])); // 1
```

SEE ALSO

Forge Standard Library

Std Storage

accesses

Signature

```
function accesses(
  address
)
external
returns (
  bytes32[] memory reads,
  bytes32[] memory writes
);
```

Description

Gets all storage slots that have been read (`reads`) or written to (`writes`) on an address.

Note that `record` must be called first.

Note

Every write also counts as an additional read.

Examples

```
/// contract NumsContract {
///     uint256 public num1 = 100; // slot 0
///     uint256 public num2 = 200; // slot 1
/// }

vm.record();
numsContract.num2();
(bytes32[] memory reads, bytes32[] memory writes) = vm.accesses(
  address(numsContract)
);
emit log_uint(uint256(reads[0])); // 1
```

recordLogs

Signature

```
function recordLogs() external;
```

Description

Tells the VM to start recording all the emitted events. To access them, use `getRecordedLogs`.

Examples

```
/// event LogCompleted(
///   uint256 indexed topic1,
///   bytes data
/// );

vm.recordLogs();

emit LogCompleted(10, "operation completed");

Vm.Log[] memory entries = vm.getRecordedLogs();

assertEq(entries.length, 1);
assertEq(entries[0].topics[0], keccak256("LogCompleted(uint256,bytes)"));
assertEq(entries[0].topics[1], bytes32(uint256(10)));
assertEq(abi.decode(entries[0].data, (string)), "operation completed");
```

getRecordedLogs

Signature

```
struct Log {
  bytes32[] topics;
  bytes data;
  address emitter;
}

function getRecordedLogs()
external
returns (
  Log[] memory
);
```

Description

Gets the emitted events recorded by `recordLogs`.

This function will consume the recorded logs when called.

Examples

```
/// event LogTopic1(
///   uint256 indexed topic1,
///   bytes data
/// );

/// event LogTopic12(
///   uint256 indexed topic1,
///   uint256 indexed topic2,
///   bytes data
/// );

/// bytes memory testData0 = "Some data";
/// bytes memory testData1 = "Other data";


// Start the recorder
vm.recordLogs();

emit LogTopic1(10, testData0);
emit LogTopic12(20, 30, testData1);

// Notice that your entries are <Interface>.Log[]
// as opposed to <instance>.Log[]
Vm.Log[] memory entries = vm.getRecordedLogs();

assertEq(entries.length, 2);

// Recall that topics[0] is the event signature
assertEq(entries[0].topics.length, 2);
assertEq(entries[0].topics[0], keccak256("LogTopic1(uint256,bytes)"));
assertEq(entries[0].topics[1], bytes32(uint256(10)));
// assertEq won't compare bytes variables. Try with strings instead.
assertEq(abi.decode(entries[0].data, (string)), string(testData0));

assertEq(entries[1].topics.length, 3);
assertEq(entries[1].topics[0], keccak256("LogTopic12(uint256,uint256,bytes)"));
assertEq(entries[1].topics[1], bytes32(uint256(20)));
assertEq(entries[1].topics[2], bytes32(uint256(30)));
assertEq(abi.decode(entries[1].data, (string)), string(testData1));

// Emit another event
emit LogTopic1(40, testData0);

// Your last read consumed the recorded logs,
// you will only get the latest emitted even after that call
entries = vm.getRecordedLogs();

assertEq(entries.length, 1);

assertEq(entries[0].topics.length, 2);
assertEq(entries[0].topics[0], keccak256("LogTopic1(uint256,bytes)"));
assertEq(entries[0].topics[1], bytes32(uint256(40)));
assertEq(abi.decode(entries[0].data, (string)), string(testData0));
```

setNonce

Signature

```
function setNonce(address account, uint64 nonce) external;
```

Description

Sets the nonce of the given account.

The new nonce must be higher than the current nonce of the account.

Examples

```
vm.setNonce(address(100), 1234);
```

getNonce

Signature

```
function getNonce(address account) external returns (uint64);
```

```
function getNonce(Wallet memory wallet) external returns (uint64);
```

Description

Gets the nonce of the given account or Wallet.

Examples

address

```
uint256 nonce = vm.getNonce(address(100));
emit log_uint(nonce); // 0
```

Wallet

```
Wallet memory alice = vm.createWallet("alice");
uint256 nonce = vm.getNonce(nonce);
emit log_uint(nonce); // 0
```

mockCall

Signature

```
function mockCall(address where, bytes calldata data, bytes calldata retdata)
external;
```

```
function mockCall(
    address where,
    uint256 value,
    bytes calldata data,
    bytes calldata retdata
) external;
```

Description

Mocks all calls to an address `where` if the call data either strictly or loosely matches `data` and returns `retdata`.

When a call is made to `where` the call data is first checked to see if it matches in its entirety with `data`. If not, the call data is checked to see if there is a partial match, with the match starting at the first byte of the call data.

If a match is found, then `retdata` is returned from the call.

Using the second signature we can mock the calls with a specific `msg.value`. `Calldata` match takes precedence over `msg.value` in case of ambiguity.

Mocked calls are in effect until `clearMockedCalls` is called.

ℹ️ Note

Calls to mocked addresses may revert if there is no code on the address. This is because Solidity inserts an `extcodesize` check before some contract calls.

To circumvent this, use the `etch` cheatcode if the mocked address has no code.

ℹ️ Internal calls

This cheatcode does not currently work on internal calls. See issue #432.

Examples

Mocking an exact call:

```
function testMockCall() public {
    vm.mockCall(
        address(0),
        abi.encodeWithSelector(MyToken.balanceOf.selector, address(1)),
        abi.encode(10)
    );
    assertEq(IERC20(address(0)).balanceOf(address(1)), 10);
}
```

Mocking an entire function:

```
function testMockCall() public {
    vm.mockCall(
        address(0),
        abi.encodeWithSelector(MyToken.balanceOf.selector),
        abi.encode(10)
    );
    assertEq(IERC20(address(0)).balanceOf(address(1)), 10);
    assertEq(IERC20(address(0)).balanceOf(address(2)), 10);
}
```

Mocking a call with a given `msg.value`:

```
function testMockCall() public {
    assertEq(example.pay{value: 10}(1), 1);
    assertEq(example.pay{value: 1}(2), 2);
    vm.mockCall(
        address(example),
        10,
        abi.encodeWithSelector(example.pay.selector),
        abi.encode(99)
    );
    assertEq(example.pay{value: 10}(1), 99);
    assertEq(example.pay{value: 1}(2), 2);
}
```

Mocking a public variable:

```
contract Example {
    uint256 public number = 10;
}

contract ExampleTest is Test {
    Example public example;

    function setUp() public {
        example = new Example();
    }

    function testMockPublicVariable() public {
        assertEq(example.number(), 10);
        vm.mockCall(
            address(example),
            abi.encodeWithSelector(bytes4(keccak256("number()"))),
            abi.encode(5)
        );
        assertEq(example.number(), 5);
    }
}
```

mockCallRevert

Signature

```
function mockCallRevert(address where, bytes calldata data, bytes calldata
retdata) external;
```

```
function mockCallRevert(
    address where,
    uint256 value,
    bytes calldata data,
    bytes calldata retdata
) external;
```

Description

Reverts all calls to an address `where` if the call data either strictly or loosely matches `data` and returns `retdata`.

`retdata` can be a raw return message or a custom error.

When a call is made to `where` the call data is first checked to see if it matches in its entirety with `data`. If not, the call data is checked to see if there is a partial match, with the match starting at the first byte of the call data.

If a match is found, then the call is reverted and `retdata` is returned.

Using the second signature we can mock the calls with a specific `msg.value`. `Calldata` match takes precedence over `msg.value` in case of ambiguity.

Reverted mock calls are in effect until `clearMockedCalls` is called.

ℹ️ **Internal calls**

This cheatcode does not currently work on internal calls. See issue #432.

Examples

Reverting an exact call with a raw error message:

```
function testMockCallRevert() public {
    vm.mockCallRevert(
        address(0),
        abi.encodeWithSelector(MyToken.balanceOf.selector, address(1)),
        "REVERT_MESSAGE"
    );
    vm.expectRevert("REVERT_MESSAGE");
    IERC20(address(0)).balanceOf(address(1));
}
```

Reverting a call with a custom error:

```
function testMockCallRevertWithCustomError() public {
    bytes memory customError = abi.encodeWithSelector(TestError.selector,
"ERROR_MESSAGE");
    vm.mockCallRevert(
        address(0),
        abi.encodeWithSelector(MyToken.balanceOf.selector, address(1)),
        customError
    );
    vm.expectRevert(customError);
    IERC20(address(0)).balanceOf(address(1));
}
```

Mocking a call with a given `msg.value`:

```
function testMockCallRevertWithValue() public {
    assertEq(example.pay{value: 10}(1), 1);
    assertEq(example.pay{value: 1}(2), 2);
    vm.mockCallRevert(
        address(example),
        10,
        abi.encodeWithSelector(example.pay.selector),
        "ERROR_MESSAGE"
    );
    assertEq(example.pay{value: 1}(2), 2);
    vm.expectRevert("ERROR_MESSAGE");
    assertEq(example.pay{value: 10}(1), 99);
}
```

clearMockedCalls

Signature

```
function clearMockedCalls() external;
```

Description

Clears all mocked calls.

coinbase

Signature

```
function coinbase(address) external;
```

Description

Sets `block.coinbase`.

Examples

```
emit log_address(block.coinbase); // 0x0000000000000000000000000000000000000000
vm.coinbase(0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8);
emit log_address(block.coinbase); // 0xea674fdde714fd979de3edf0f56aa9716b898ec8
```

broadcast

Signature

```
function broadcast() external;

function broadcast(address who) external;

function broadcast(uint256 privateKey) external;
```

Description

Using the address that calls the test contract or the address / private key provided as the sender, has the next call (at this call depth only and excluding cheatcode calls) create a transaction that can later be signed and sent onchain.

Examples

```
function deploy() public {
    cheats.broadcast(ACCOUNT_A);
    Test test = new Test();

    // this won't generate tx to sign
    uint256 b = test.t(4);

    // this will
    cheats.broadcast(ACCOUNT_B);
    test.t(2);

    // this also will, using a private key from your environment variables
    cheats.broadcast(vm.envUint("PRIVATE_KEY"));
    test.t(3);
}
```

SEE ALSO

- startBroadcast
- stopBroadcast

startBroadcast

Signature

```
function startBroadcast() external;
```

```
function startBroadcast(address who) external;
```

```
function startBroadcast(uint256 privateKey) external;
```

Description

Using the address that calls the test contract or the address / private key provided as the sender, has all subsequent calls (at this call depth only and excluding cheatcode calls) create transactions that can later be signed and sent onchain.

Examples

```
function t(uint256 a) public returns (uint256) {
    uint256 b = 0;
    emit log_string("here");
    return b;
}

function deployOther() public {
    vm.startBroadcast(ACCOUNT_A);
    Test test = new Test();

    // will trigger a transaction
    test.t(1);

    vm.stopBroadcast();

    // broadcast again, this time using a private key from your environment
variables
    vm.startBroadcast(vm.envUint("PRIVATE_KEY"));
    test.t(3);
    vm.stopBroadcast();
}
```

SEE ALSO

- broadcast
- stopBroadcast

stopBroadcast

Signature

```
function stopBroadcast() external;
```

Description

Stops collecting transactions for later on-chain broadcasting.

Examples

```
function deployNoArgs() public {
    // broadcast the next call
    cheats.broadcast();
    Test test1 = new Test();

    // broadcast all calls between this line and `stopBroadcast`
    cheats.startBroadcast();
    Test test2 = new Test();
    cheats.stopBroadcast();
}
```

pauseGasMetering

Signature

```
function pauseGasMetering() external;
```

Description

Pauses gas metering (i.e. `gasleft()` does not decrease as operations are executed).

This can be useful for getting a better sense of gas costs, by turning off gas metering for unnecessary code, as well as long-running scripts that would otherwise run out of gas.

ℹ️ Note

`pauseGasMetering` *turns off DoS protections* that come from metering gas usage.

Exposing a service that assumes a particular instance of the EVM will complete due to gas usage no longer is true, and a timeout should be enabled in that case.

`resumeGasMetering`

Signature

```
function resumeGasMetering() external;
```

Description

Resumes gas metering (i.e. `gasleft()` will decrease as operations are executed). Gas usage will resume at the same amount at which it was paused.

txGasPrice

Signature

```
function txGasPrice(uint256) external;
```

```
function txGasPrice(uint256 newGasPrice) external;
```

Description

Sets `tx.gasprice` **for the rest of the transaction**.

Examples

We can use this to get accurate gas usage for a transaction.

```
function testCalculateGas() public {
    vm.txGasPrice(2);
    uint256 gasStart = gasleft();
    myContract.doStuff();
    uint256 gasEnd = gasleft();
    uint256 gasUsed = (gasStart - gasEnd) * tx.gasprice; // tx.gasprice is now
2
}
```

`startStateDiffRecording`

Signature

```
function startStateDiffRecording()
```

Description

Records all state changes as part of CREATE, CALL or SELFDESTRUCT opcodes in order, along with the context of the calls. Refer to `stopAndReturnStateDiff` for more details on how to access and interpret the recorded state changes.

stopAndReturnStateDiff

Signature

```
enum AccountAccessKind {
    Call,
    DelegateCall,
    CallCode,
    StaticCall,
    Create,
    SelfDestruct,
    Resume,
    Balance,
    Extcodesize,
    Extcodehash,
    Extcodecopy
}

struct ChainInfo {
    uint256 forkId;
    uint256 chainId;
}

struct AccountAccess {
    ChainInfo chainInfo;
    AccountAccessKind kind;
    address account;
    address accessor;
    bool initialized;
    uint256 oldBalance;
    uint256 newBalance;
    bytes deployedCode;
    uint256 value;
    bytes data;
    bool reverted;
    StorageAccess[] storageAccesses;
}

struct StorageAccess {
    address account;
    bytes32 slot;
    bool isWrite;
    bytes32 previousValue;
    bytes32 newValue;
    bool reverted;
}

function stopAndReturnStateDiff() external returns (AccountAccess[] memory
accesses);
```

Description

Retrieves state changes recorded after a call to `startStateDiffRecording`. This function will consume the recorded state diffs when called and disable state diff recording. One may call `startStateDiffRecording` to resume recording.

There are two types of state change records; account accesses and storage accesses represented as `AccountAccess` and `StorageAccess`.

Account state changes (`AccountAccess`) are recorded at the start of a new EVM context; i.e. induced by the various CREATE, CALL and SELFDESTRUCT operations. An `AccountAccess` record contain storage accesses, represented as `StorageAccess`, that occurred before it was preempted via sub-calls or create operations.

The ordering of `AccountAccess` records reflect the EVM execution order of their associated operations. An `AccountAccess` is created whenever an EVM context is created or resumed. If a sub-context is created, a `Resume` `AccountAccess` is recorded to indicate that a previous `AccountAccess` that was pre-empted has been resumed.

`AccountAccessKind`

The kind of account access that determines the `account` that was accessed. This is typically designated by the EVM operation that initiated the account's execution context. If kind is `Call`, `DelegateCall`, `StaticCall` or `CallCode`, then the `account` is the callee. If kind is Create, then the account is the newly created account. If kind is SelfDestruct, then the account is the selfdestruct recipient. If kind is a Resume, then account represents an execution context that had resumed.

- `Call` - The account was called
- `DelegateCall` - The account was called via delegate call
- `CallCode` - The account was called via callcode
- `StaticCall` - The account was called via staticcall
- `Create` - The account was created
- `SelfDestruct` - The account was selfdestructed
- `Resume` - Indicates that a previously pre-emptyed account access was resumed
- `Balance` - The account's codesize was read
- `Extcodesize` - The account's codesize was read
- `Extcodehash` - The account's codehash was read
- `Extcodecopy` - The account's code was copied

`AccountAccess`

- `chainInfo` - The chain and fork the accessed occurred.

- `kind` - The kind of account access. This determines how to interpret the `AccountAccess`
- `account` - The account that was accessed. It's the account created for `AccountAccessKind.Create`. In the case of an `AccountAccessKind.SelfDestruct`, it's the selfdestruct recipient. For all other types of `AccountAccessKind`, it's the account of the current EVM context.
- `accessor` - What accessed `account`. That is either the account creator, caller or the account being selfdestructed.
- `initialized` - If the account was initialized or empty prior to the access. An account is considered initialized if it has code, a non-zero nonce, or a non-zero balance.
- `oldBalance`: The previous balance of the accessed `account`.
- `newBalance` - The potential new balance of the accessed account. That is, all balance changes are recorded here, even if reverts occurred.
- `deployedCode` - Code of the `account` deployed in the case of `AccountAccessKind.Create`. This field is empty For all other account access kinds.
- `value` - The value passed along with the account access.
- `data` - Input data provided (i.e. `msg.data`) in the case of a `CREATE` or `CALL` type access.
- `reverted` - If this access reverted in either the current or parent context.
- `storageAccesses` - An ordered list of storage accesses made while the account access is non-preemptive.

`StorageAccess`

The storage accesses made during an `AccountAccess`. `StorageAccess` cannot exist without an associated `AccountAccess`. This means that when state diffs begins on the given context, storage accesses made during that context are not recorded as the context (but not its sub-contexts) isn't recorded.

`StorageAccess` contains the following fields:

- `account` - A account whose storage was accessed
- `slot` - The slot that was accessed
- `isWrite` - If the access was a write
- `previousValue` - The value of the slot prior to this storage access
- `newValue` - The value of the slot after this storage access
- `reverted` - If this access was reverted

Resumed `AccountAccess`

This type of AccountAccess is generated when a sub-context returns to its parent context. It retains the same values as the original context, including `accessor`, `account`,

`initialized`, `storageAccesses`, and `reverted`. The following control flow table illustrate how Resume AccountAccesses are recorded.

Step in Contract A's `alpha()`	Step in Contract B's `beta()`	AccountAccess Records State
Call A.alpha()		[A.call]
Access state		[A.call[A.access]]
Call B.beta()	B.beta() begins	[A.call[A.access], B.call]
(Execution Paused)	Access state	[A.call[A.access], B.call[B.access]]
	Return	
Resume execution	(Return to A.alpha())	[A.call[A.access], B.call[B.access]]
Access state		[A.call[A.access], B.call[B.access], A.resume[A.access']]

ℹ️ Note

A Resumed AccountAccess is created only if storage accesses occurred after a context was resumed.

Example: Recording storage state changes during a CREATE operation

```
contract Contract {
    uint256 internal _reserved;
    uint256 public data;
    constructor(uint _data) payable { data = _data; }
}

vm.startStateDiffRecording();
Contract contract = new Contract{value: 1 ether}(100);
Vm.AccountAccess[] memory records = vm.stopAndReturnStateDiff();

assertEq(records.length, 1);
assertEq(records[0].kind, Vm.AccountAccessKind.Create);
assertEq(records[0].account, address(contract));
assertEq(records[0].accessor, address(this));
assertEq(records[0].initialized, true);
assertEq(records[0].oldBalance, 0);
assertEq(records[0].newBalance, 1 ether);
assertEq(records[0].deployedCode, address(contract).code);
assertEq(records[0].value, 1 ether);
assertEq(records[0].data, abi.encodePacked(type(Contract).creationCode,
(uint(100))));
assertEq(records[0].reverted, false);

assertEq(records[0].storageAccesses.length, 1);
assertEq(records[0].storageAccesses[0].account, address(contract));
assertEq(records[0].storageAccesses[0].slot, bytes32(uint256(1)));
assertEq(records[0].storageAccesses[0].isWrite, true);
assertEq(records[0].storageAccesses[0].previousValue, bytes32(uint(0)));
assertEq(records[0].storageAccesses[0].newValue, bytes32(uint(100)));
assertEq(records[0].storageAccesses[0].reverted, false);
```

Note that there are no `Resume` account accesses in this example.

Example: Resumed Account Access

```
contract Foo {
    Bar b;
    uint256 public val;
    constructor(Bar _b) { b = _b; }
    function run() external {
        val = val + 1;
        b.run();
        val = val + 1;
    }
}
contract Bar {
    function run() external {}
}

Bar bar = new Bar();
Foo foo = new Foo(bar);

vm.startStateDiffRecording();
foo.run();
Vm.AccountAccess[] memory records = vm.stopAndReturnStateDiff();

assertEq(records.length, 3);
Vm.AccountAccess memory fooCall = records[0];
assertEq(fooCall.kind, Vm.AccountAccessKind.Call);
assertEq(fooCall.account, address(foo));
assertEq(fooCall.accessor, address(this));
// foo.val increment
assertEq(fooCall.storageAccesses.length, 2);
assertEq(fooCall.storageAccesses[0].isWrite, false);
assertEq(fooCall.storageAccesses[1].isWrite, true);
assertEq(fooCall.storageAccesses[1].oldValue, bytes32(uint(0)));
assertEq(fooCall.storageAccesses[1].newValue, bytes32(uint(1)));

// bar.run CALL
Vm.AccountAccess memory barCall = records[1];
assertEq(barCall.kind, Vm.AccountAccessKind.Call);
assertEq(barCall.account, address(bar));
assertEq(barCall.accessor, address(foo));

// foo.run RESUME
Vm.AccountAccess memory fooResume = records[2];
assertEq(fooResume.kind, Vm.AccountAccessKind.Resume);
// foo.val increment
assertEq(fooResume.storageAccesses.length, 2);
assertEq(fooResume.storageAccesses[0].isWrite, false);
assertEq(fooResume.storageAccesses[1].isWrite, true);
assertEq(fooResume.storageAccesses[1].oldValue, bytes32(uint(1)));
assertEq(fooResume.storageAccesses[1].newValue, bytes32(uint(2)));
```

Assertions

- `expectRevert`
- `expectEmit`
- `expectCall`

expectRevert

Signature

```
function expectRevert() external;

function expectRevert(bytes4 message) external;

function expectRevert(bytes calldata message) external;
```

Description

If the **next call** does not revert with the expected data `message`, then `expectRevert` will.

After calling `expectRevert`, calls to other cheatcodes before the reverting call are ignored.

This means, for example, we can call `prank` immediately before the reverting call.

There are 3 signatures:

- **Without parameters**: Asserts that the next call reverts, regardless of the message.
- **With `bytes4`**: Asserts that the next call reverts with the specified 4 bytes.
- **With `bytes`**: Asserts that the next call reverts with the specified bytes.

⚠️ **Gotcha: Usage with low-level calls**

Normally, a call that succeeds returns a status of `true` (along with any return data) and a call that reverts returns `false`.

The Solidity compiler will insert checks that ensures that the call succeeded, and revert if it did not.

On low level calls, the `expectRevert` cheatcode works by making the `status` boolean returned by the low level call correspond to whether the `expectRevert` succeeded or not, NOT whether or not the low-level call succeeds. Therefore, `status` being false corresponds to the cheatcode failing.

Apart from this, `expectRevert` also mangles return data on low level calls, and is not usable.

See the following example. For clarity, `status` has been renamed to `revertsAsExpected`:

```
function testLowLevelCallRevert() public {
    vm.expectRevert(bytes("error message"));
    (bool revertsAsExpected, ) = address(myContract).call(myCalldata);
    assertTrue(revertsAsExpected, "expectRevert: call did not revert");
}
```

Examples

To use `expectRevert` with a `string`, pass it as a string literal.

```
vm.expectRevert("error message");
```

To use `expectRevert` with a custom error type without parameters, use its selector.

```
vm.expectRevert(CustomError.selector);
```

To use `expectRevert` with a custom error type with parameters, ABI encode the error type.

```
vm.expectRevert(
    abi.encodeWithSelector(CustomError.selector, 1, 2)
);
```

If you need to assert that a function reverts *without* a message, you can do so with `expectRevert(bytes(""))`.

```
function testExpectRevertNoReason() public {
    Reverter reverter = new Reverter();
    vm.expectRevert(bytes(""));
    reverter.revertWithoutReason();
}
```

Message-less reverts happen when there is an EVM error, such as when the transaction consumes more than the block's gas limit.

If you need to assert that a function reverts a four character message, e.g. `AAAA`, you can do so with:

```
function testFourLetterMessage() public {
    vm.expectRevert(bytes("AAAA"));
}
```

If used `expectRevert("AAAA")`, the compiler would throw an error because it wouldn't know which overload to use.

Finally, you can also have multiple `expectRevert()` checks in a single test.

```
function testMultipleExpectReverts() public {
    vm.expectRevert("INVALID_AMOUNT");
    vault.send(user, 0);

    vm.expectRevert("INVALID_ADDRESS");
    vault.send(address(0), 200);
}
```

SEE ALSO

Forge Standard Library

Std Errors

expectEmit

Signature

```
function expectEmit() external;
```

```
function expectEmit(
    bool checkTopic1,
    bool checkTopic2,
    bool checkTopic3,
    bool checkData
) external;
```

```
function expectEmit(address emitter) external;
```

```
function expectEmit(
    bool checkTopic1,
    bool checkTopic2,
    bool checkTopic3,
    bool checkData,
    address emitter
) external;
```

Description

Assert a specific log is emitted during the next call.

1. Call the cheat code, specifying whether we should check the first, second or third topic, and the log data (`expectEmit()` checks them all). Topic 0 is always checked.
2. Emit the event we are supposed to see during the next call.
3. Perform the call.

You can perform steps 1 and 2 multiple times to match a *sequence* of events in the next call.

If the event is not available in the current scope (e.g. if we are using an interface, or an external smart contract), we can define the event ourselves with an identical event signature.

There are 2 varieties of `expectEmit`:

- **Without checking the emitter address**: Asserts the topics match **without** checking the emitting address.

- **With `address`** : Asserts the topics match and that the emitting address matches.

ℹ️ Matching sequences

In functions that emit a lot of events, it's possible to "skip" events and only match a specific sequence, but this sequence must always be in order. As an example, let's say a function emits events: `A, B, C, D, E, F, F, G`.

`expectEmit` will be able to match ranges with and without skipping events in between:

- `[A, B, C]` is valid.
- `[B, D, F]` is valid.
- `[G]` or any other single event combination is valid.
- `[B, A]` or similar out-of-order combinations are **invalid** (events must be in order).
- `[C, F, F]` is valid.
- `[F, F, C]` is **invalid** (out of order).

Examples

This does not check the emitting address.

```
event Transfer(address indexed from, address indexed to, uint256 amount);

function testERC20EmitsTransfer() public {
    vm.expectEmit();

    // We emit the event we expect to see.
    emit MyToken.Transfer(address(this), address(1), 10);

    // We perform the call.
    myToken.transfer(address(1), 10);
}
```

This does check the emitting address.

```
event Transfer(address indexed from, address indexed to, uint256 amount);

function testERC20EmitsTransfer() public {
    // We check that the token is the event emitter by passing the address.
    vm.expectEmit(address(myToken));
    emit MyToken.Transfer(address(this), address(1), 10);

    // We perform the call.
    myToken.transfer(address(1), 10);
}
```

We can also assert that multiple events are emitted in a single call.

```
function testERC20EmitsBatchTransfer() public {
    // We declare multiple expected transfer events
    for (uint256 i = 0; i < users.length; i++) {
        // Here we use the longer signature for demonstration purposes. This
call checks
        // topic0 (always checked), topic1 (true), topic2 (true), NOT topic3
(false), and data (true).
        vm.expectEmit(true, true, false, true);
        emit Transfer(address(this), users[i], 10);
    }

    // We also expect a custom `BatchTransfer(uint256 numberOfTransfers)`
event.
    vm.expectEmit(false, false, false, true);
    emit BatchTransfer(users.length);

    // We perform the call.
    myToken.batchTransfer(users, 10);
}
```

This example fails, as the expected event is not emitted on the next call.

```
event Transfer(address indexed from, address indexed to, uint256 amount);

function testERC20EmitsTransfer() public {
    // We check that the token is the event emitter by passing the address as
the fifth argument.
    vm.expectEmit(true, true, false, true, address(myToken));
    emit MyToken.Transfer(address(this), address(1), 10);

    // We perform an unrelated call that won't emit the intended event,
    // making the cheatcode fail.
    myToken.approve(address(this), 1e18);
    // We perform the call, but it will have no effect as the cheatcode has
already failed.
    myToken.transfer(address(1), 10);
}
```

expectCall

```
function expectCall(address callee, bytes calldata data) external;
```

```
function expectCall(address callee, bytes calldata data, uint64 count)
external;
```

```
function expectCall(
    address callee,
    uint256 value,
    bytes calldata data
) external;
```

```
function expectCall(
    address callee,
    uint256 value,
    bytes calldata data,
    uint64 count
) external;
```

Description

Expects that a call to a specified address `callee`, where the call data either strictly or loosely matches `data`. The cheatcode can be called in two ways:

- If no `count` parameter is specified, the call will be expected to be made at least the amount of times the cheatcode was called. For the same calldata, you cannot call the cheatcode with no `count` and then pass in a `count` parameter.
- If `count` is specified, the call will be expected to be made strictly `count` times. For the same calldata, the `count` value cannot be overwritten with another cheatcode call, nor it can be increment by calling the cheatcode without a `count` parameter.

`count` can also be set to 0 to assert that a call is not made.

When a call is made to `callee` the call data is first checked to see if it matches in its entirety with `data`. If not, the call data is checked to see if there is a partial match, with the match starting at the first byte of the call data.

Using the second signature we can also check if the call was made with the expected `msg.value`.

If the test terminates without the call being made, the test fails.

> ℹ️ **Internal calls**
>
> This cheatcode does not currently work on internal calls. See issue #432.

Examples

Expect that `transfer` is called on a token `MyToken` one time:

```
address alice = makeAddr("alice");
emit log_address(alice);
vm.expectCall(
  address(token), abi.encodeCall(token.transfer, (alice, 10))
);
token.transfer(alice, 10);
// [PASS]
```

Expect that `transfer` is called on a token `MyToken` *at least* two times:

```
address alice = makeAddr("alice");
emit log_address(alice);
vm.expectCall(
  address(token), abi.encodeCall(token.transfer, (alice, 10))
);
vm.expectCall(
  address(token), abi.encodeCall(token.transfer, (alice, 10))
);
token.transfer(alice, 10);
token.transfer(alice, 10);
token.transfer(alice, 10);
// [PASS]
```

Expect that `transfer` is not called on a token `MyToken`:

```
address alice = makeAddr("alice");
emit log_address(alice);
vm.expectCall(
  address(token), abi.encodeCall(token.transfer, (alice, 10)), 0
);
token.transferFrom(alice, address(0), 10);
// [PASS]
```

Expect that `transfer` with any calldata is called on a token `MyToken` 2 times:

```
address alice = makeAddr("alice");
emit log_address(alice);
vm.expectCall(
  address(token), abi.encodeWithSelector(token.transfer.selector), 2
);
token.transfer(alice, 10);
token.transfer(alice, 10);
// [PASS]
```

Expect that `pay` is called on a `Contract` with a specific `msg.value` and `calldata`:

```
Contract target = new Contract();
vm.expectCall(
            address(target),
            1,
            abi.encodeWithSelector(target.pay.selector, 2)
        );
target.pay{value: 1}(2);
// [PASS]
```

Expect that `pay` is called on a `Contract` with a specific `msg.value` and `calldata` 3 times:

```
Contract target = new Contract();
vm.expectCall(
            address(target),
            1,
            abi.encodeWithSelector(target.pay.selector, 2),
            3
        );
target.pay{value: 1}(2);
target.pay{value: 1}(2);
target.pay{value: 1}(2);
// [PASS]
```

Fuzzer

- `assume`

assume

Signature

```
function assume(bool) external;
```

Description

If the boolean expression evaluates to false, the fuzzer will discard the current fuzz inputs and start a new fuzz run.

The `assume` cheatcode should mainly be used for very narrow checks. Broad checks will slow down tests as it will take a while to find valid values, and the test may fail if you hit the max number of rejects.

You can configure the rejection thresholds by setting `fuzz.max_test_rejects` in your `foundry.toml` file.

For broad checks, such as ensuring a `uint256` falls within a certain range, you can bound your input with the modulo operator or Forge Standard's `bound` method.

More information on filtering via `assume` can be found here.

Examples

```
// Good example of using assume
function testSomething(uint256 a) public {
    vm.assume(a != 1);
    require(a != 1);
    // [PASS]
}
```

```
// In this case assume is not a great fit, so you should bound inputs manually
function testSomethingElse(uint256 a) public {
    a = bound(a, 100, 1e36);
    require(a >= 100 && a <= 1e36);
    // [PASS]
}
```

SEE ALSO

Forge Standard Library

bound

Forking

- `createFork`
- `selectFork`
- `createSelectFork`
- `activeFork`
- `rollFork`
- `makePersistent`
- `revokePersistent`
- `isPersistent`
- `allowCheatcodes`
- `transact`

createFork

Signature

```
// Creates a new fork with the given endpoint and the _latest_ block and
returns the identifier of the fork
function createFork(string calldata urlOrAlias) external returns (uint256)
```

```
// Creates a new fork with the given endpoint and block and returns the
identifier of the fork
function createFork(string calldata urlOrAlias, uint256 block) external returns
(uint256);
```

```
// Creates a new fork with the given endpoint and at the block the given
transaction was mined in, and replays all transaction mined in the block before
the transaction
function createFork(string calldata urlOrAlias, bytes32 transaction) external
returns (uint256);
```

Description

Creates a new fork from the given endpoint and returns the identifier of the fork. If a block number is passed as an argument, the fork will begin on that block, otherwise it will begin on the *latest* block.

If a transaction hash is provided, it will roll the fork to the block the transaction was mined in and replays all previously executed transactions.

Examples

Create a new mainnet fork with the latest block number:

```
uint256 forkId = vm.createFork(MAINNET_RPC_URL);
vm.selectFork(forkId);

assertEq(block.number, 15_171_037); // as of time of writing, 2022-07-19
04:55:27 UTC
```

Create a new mainnet fork with a given block number:

```
uint256 forkId = vm.createFork(MAINNET_RPC_URL, 1_337_000);
vm.selectFork(forkId);

assertEq(block.number, 1_337_000);
```

SEE ALSO

- activeFork
- selectFork
- createSelectFork

selectFork

Signature

```
function selectFork(uint256 forkId) external;
```

Description

Takes a fork identifier created by `createFork` and sets the corresponding forked state as active.

Examples

Select a previously created fork:

```
uint256 forkId = vm.createFork(MAINNET_RPC_URL);

vm.selectFork(forkId);

assertEq(vm.activeFork(), forkId);
```

SEE ALSO

- createFork
- activeFork

createSelectFork

Signature

```
function createSelectFork(string calldata urlOrAlias) external returns
(uint256);
```

```
function createSelectFork(string calldata urlOrAlias, uint256 block) external
returns (uint256);
```

```
function createSelectFork(string calldata urlOrAlias, bytes32 transaction)
external returns (uint256);
```

Description

Creates *and* selects a new fork from the given endpoint and returns the identifier of the fork. If a block number is passed as an argument, the fork will begin on that block, otherwise it will begin on the *latest* block.

If a transaction hash is provided, it will roll the fork to the block the transaction was mined in and replays all previously executed transactions.

Examples

Create and select a new mainnet fork with the latest block number:

```
uint256 forkId = vm.createSelectFork(MAINNET_RPC_URL);

assertEq(block.number, 15_171_037); // as of time of writing, 2022-07-19
04:55:27 UTC
```

Create and select a new mainnet fork with a given block number:

```
uint256 forkId = vm.createSelectFork(MAINNET_RPC_URL, 1_337_000);

assertEq(block.number, 1_337_000);
```

SEE ALSO

- createFork
- selectFork

activeFork

Signature

```
function activeFork() external returns (uint256);
```

Description

Returns the identifier for the currently active fork. Reverts if no fork is currently active.

Examples

Get the currently active fork id:

```
uint256 mainnetForkId = vm.createFork(MAINNET_RPC_URL);
uint256 optimismForkId = vm.createFork(OPTIMISM_RPC_URL);

assert(mainnetForkId != optimismForkId);

vm.selectFork(mainnetForkId);
assertEq(vm.activeFork(), mainnetForkId);

vm.selectFork(optimismForkId);
assertEq(vm.activeFork(), optimismForkId);
```

SEE ALSO

- createFork
- selectFork

rollFork

Signature

```
// roll the _active_ fork to the given block
function rollFork(uint256 blockNumber) external;


// roll the _active_ fork to the block in which the transaction was mined it
and replays all previously executed transactions
function rollFork(bytes32 transaction) external;


// Same as `rollFork(uint256 blockNumber)` but uses the fork corresponding to
the `forkId`
function rollFork(uint256 forkId, uint256 blockNumber) external;


// Same as `rollFork(bytes32 transaction)` but uses the fork corresponding to
the `forkId`
function rollFork(uint256 forkId, bytes32 transaction) external;
```

Description

Sets `block.number`. If a fork identifier is passed as an argument, it will update that fork, otherwise it will update the currently active fork.

If a transaction hash is provided, it will roll the fork to the block the transaction was mined in and replays all previously executed transactions.

Examples

Set `block.number` for the currently active fork:

```
uint256 forkId = vm.createFork(MAINNET_RPC_URL);
vm.selectFork(forkId);

assertEq(block.number, 15_171_037); // as of time of writing, 2022-07-19
04:55:27 UTC

vm.rollFork(15_171_057);

assertEq(block.number, 15_171_057);
```

Set `block.number` for the fork identified by the passed `forkId` argument:

```
uint256 optimismForkId = vm.createFork(OPTIMISM_RPC_URL);

vm.rollFork(optimismForkId, 1_337_000);

vm.selectFork(optimismForkId);

assertEq(block.number, 1_337_000);
```

SEE ALSO

- roll
- createFork
- selectFork
- activeFork

makePersistent

Signature

```
function makePersistent(address account) external;
function makePersistent(address account0, address account1) external;
function makePersistent(address account0, address account1, address account2)
external;
function makePersistent(address[] calldata accounts) external;
```

Description

Each fork (`createFork`) has its own independent storage, which is also replaced when another fork is selected (`selectFork`). By default, only the test contract account and the caller are persistent across forks, which means that changes to the state of the test contract (variables) are preserved when different forks are selected. This way data can be shared by storing it in the contract's variables.

However, with this cheatcode, it is possible to mark the specified accounts as persistent, which means that their state is available regardless of which fork is currently active.

Examples

Mark a new contract as persistent

```
contract SimpleStorageContract {
    string public value;

    function set(uint256 _value) public {
        value = _value;
    }
}

function testMarkPersistent() public {
    // by default the `sender` and the contract itself are persistent
    assert(cheats.isPersistent(msg.sender));
    assert(cheats.isPersistent(address(this)));

    // select a specific fork
    cheats.selectFork(mainnetFork);

    // create a new contract that's stored in the `mainnetFork` storage
    SimpleStorageContract simple = new SimpleStorageContract();

    // `simple` is not marked as persistent
    assert(!cheats.isPersistent(address(simple)));

    // contract can be used
    uint256 expectedValue = 99;
    simple.set(expectedValue);
    assertEq(simple.value(), expectedValue);

    // mark as persistent
    cheats.makePersistent(address(simple));

    // select a different fork
    cheats.selectFork(optimismFork);

    // ensure contract is still persistent
    assert(cheats.isPersistent(address(simple)));

    // value is set as expected
    assertEq(simple.value(), expectedValue);
}
```

SEE ALSO

- isPersistent
- revokePersistent
- createFork
- selectFork

`revokePersistent`

Signature

```
function revokePersistent(address) external;
function revokePersistent(address[] calldata) external;
```

Description

The counterpart of `makePersistent`, that makes the given contract not persistent across fork swaps

Examples

Revoke a persistent status of a contract

```
contract SimpleStorageContract {
    string public value;

    function set(uint256 _value) public {
        value = _value;
    }
}

function testRevokePersistent() public {
    // select a specific fork
    cheats.selectFork(mainnetFork);

    // create a new contract that's stored in the `mainnetFork` storage
    SimpleStorageContract simple = new SimpleStorageContract();

    // `simple` is not marked as persistent
    assert(!cheats.isPersistent(address(simple)));

    // make it persistent
    cheats.makePersistent(address(simple));

    // ensure it is persistent
    assert(cheats.isPersistent(address(simple)));

    // revoke it
    cheats.revokePersistent(address(simple));

    // contract no longer persistent
    assert(!cheats.isPersistent(address(simple)));
}
```

SEE ALSO

- isPersistent
- revokePersistent
- createFork
- selectFork

isPersistent

Signature

```
function isPersistent(address) external returns (bool);
```

Description

Returns whether an account is marked as persistent (`makePersistent`).

Examples

Check default status of `msg.sender` and the current test account

```
// By default the `sender` and the test contract itself are persistent
assert(cheats.isPersistent(msg.sender));
assert(cheats.isPersistent(address(this)));
```

SEE ALSO

- makePersistent
- revokePersistent

allowCheatcodes

Signature

```
function allowCheatcodes(address) external;
```

Description

In forking mode, explicitly grant the given address cheatcode access.

By default, the test contract, and its deployer are allowed to access cheatcodes. In addition to that, cheat code access is granted if the contract was deployed by an address that already has cheatcode access. This will prevent cheatcode access from accounts already deployed on the forked network.

> ℹ️ **Note**
>
> This is only useful for more complex test setups in forking mode.

---

transact

Signature

```
// Fetches the given transaction from the active fork and executes it on the
current state
function transact(bytes32 txHash) external;
// Fetches the given transaction from the given fork and executes it on the
current state
function transact(uint256 forkId, bytes32 txHash) external;
```

Description

In forking mode, fetches the Transaction from the provider and executes it on the current state

Examples

Enter forking mode and execute a transaction:

```
// Enter forking mode at block: https://etherscan.io/block/15596646
uint256 fork = vm.createFork(MAINNET_RPC_URL, 15596646);
vm.selectFork(fork);

// a random transfer transaction in the block:
https://etherscan.io/tx/0xaba74f25a17cf0d95d1c6d0085d6c83fb8c5e773ffd2573b99a95
3256f989c89
bytes32 tx =
0xaba74f25a17cf0d95d1c6d0085d6c83fb8c5e773ffd2573b99a953256f989c89;

address sender = address(0xa98218cdc4f63aCe91ddDdd24F7A580FD383865b);
address recipient = address(0x0C124046Fa7202f98E4e251B50488e34416Fc306);

assertEq(sender.balance, 5764124000000000);
assertEq(recipient.balance, 3936000000000000);

// transfer amount: 0.003936 Ether
uint256 transferAmount = 3936000000000000;

// expected balance changes once the transaction is executed
uint256 expectedRecipientBalance = recipient.balance + transferAmount;
uint256 expectedSenderBalance = sender.balance - transferAmount;

// execute the transaction
vm.transact(tx);

// recipient received transfer
assertEq(recipient.balance, expectedRecipientBalance);

// sender's balance decreased by transferAmount and gas
assert(sender.balance < expectedSenderBalance);
```

SEE ALSO

- roll
- createFork
- selectFork
- activeFork

External

- `ffi`
- `prompt`
- `projectRoot`
- `getCode`
- `getDeployedCode`
- `sleep`
- `unixTime`
- `setEnv`
- `envOr`
- `envBool`
- `envUint`
- `envInt`
- `envAddress`
- `envBytes32`
- `envString`
- `envBytes`
- `parseJson`
- `fs`

`ffi`

Signature

```
function ffi(string[] calldata) external returns (bytes memory);
```

Description

Calls an arbitrary command if `ffi` is enabled.

It is generally advised to use this cheat code as a last resort, and to not enable it by default, as anyone who can change the tests of a project will be able to execute arbitrary commands on devices that run the tests.

Tips

- By default the `ffi` cheatcode assumes the output of the command is a hex encoded value (e.g. a hex string of an ABI encoded value). If hex decoding fails, it will return the output as UTF8 bytes that you can cast to a string.
- Make sure that the output does not include a `\n` newline character. (e.g in Rust use `print! vs println!`)
- Remember that the script will be executed from the top-level directory of your project, not inside `test`
- Make sure that the inputs array does not have empty elements. They will be handled as inputs by the script, instead of space
- Use the cheatcode `toString` to easily convert arbitrary data to strings, so that you can pass them as command-line arguments

Examples

ABI encoded output

```
string[] memory inputs = new string[](3);
inputs[0] = "echo";
inputs[1] = "-n";
// ABI encoded "gm", as a hex string
inputs[2] =
"0x0000000000000000000000000000000000000000000000000000000000000020000000000000000
0000000000000000000000000000000000000000000000000002676d000000000000000000000000
00000000000000000000000000000000000000";

bytes memory res = vm.ffi(inputs);
string memory output = abi.decode(res, (string));
assertEq(output, "gm");
```

UTF8 string output

```
string[] memory inputs = new string[](3);
inputs[0] = "echo";
inputs[1] = "-n";
inputs[2] = "gm";

bytes memory res = vm.ffi(inputs);
assertEq(string(res), "gm");
```

prompt

Signature

```
function prompt(string calldata promptText) external returns (string memory
input);
function promptSecret(string calldata promptText) external returns (string
memory input);
```

Description

Display an interactive prompt to the user for inserting arbitrary data.

`vm.prompt` displays an interactive input, while `vm.promptSecret` displays a hidden input, used for passwords and other secret information that should not leak to the terminal.

> ℹ️ **Note**
>
> This cheatcode is meant to be used in scripts — not tests. It is also advised to follow the best practices below for testing scripts that use `vm.prompt` and handling timeouts, since scripts might otherwise hang or revert. This cheatcode reverts when running in a non-interactive shell.

Configuration

In order to prevent unwanted hangups, `vm.prompt` has a timeout configuration.

In your `foundry.toml`:

```
prompt_timeout = 120
```

Default value is `120` and values are in seconds.

Best practices

Testing scripts that use `vm.prompt`

When testing scripts containing `vm.prompt` it is recommended to use the following pattern:

```
contract Script {
    function run() public {
        uint256 myUint = vm.parseUint(vm.prompt("enter uint"));
        run(myUint);
    }

    function run(uint256 myUint) public {
        // actual logic
    }
}
```

That way, we are keeping the UX gain (don't have to provide `--sig` argument when running the script), but tests can set any value to `myUint` and not just a hardcoded default.

Handling timeouts

When a user fails to provide an input before the timeout expires, the `vm.prompt` cheatcode reverts. If you'd like, timeouts can be handled by using `try/catch`:

```
string memory input;

try vm.prompt("Username") returns (string memory res) {
    input = res;
}
catch (bytes memory) {
    input = "Anonymous";
}
```

Examples

Choose RPC endpoint

Provide an option to choose the RPC/chain to run on.

In your `foundry.toml` file:

```
[rpc_endpoints]
mainnet = "https://eth.llamarpc.com"
polygon = "https://polygon.llamarpc.com"
```

In your script:

```
string memory rpcEndpoint = vm.prompt("RPC endpoint");
vm.createSelectFork(rpcEndpoint);
```

Parse user input into native types

We can use the string parsing cheatcodes to parse the responses from users:

```
uint privateKey = uint(vm.parseBytes32(vm.promptSecret("Private key")));
address to = vm.parseAddress(vm.prompt("Send to"));
uint amount = vm.parseUint(vm.prompt("Amount (wei)"));
vm.broadcast(privateKey);
payable(to).transfer(amount);
```

projectRoot

Signature

```
function projectRoot() external returns (string memory);
```

Description

Returns the root directory of the current Foundry project.

getCode

Signature

```
function getCode(string calldata) external returns (bytes memory);
```

Description

Returns the **creation** bytecode for a contract in the project given the path to the contract.

The calldata parameter can either be in the form `ContractFile.sol` (if the filename and contract name are the same), `ContractFile.sol:ContractName`, or the path to an artifact, relative to the root of your project.

ℹ️ Note

`getCode` requires read permission for the output directory, see file cheatcodes.

To grant read access set `fs_permissions = [{ access = "read", path = "./out"}]` in your `foundry.toml`.

Examples

```
MyContract myContract = new MyContract(arg1, arg2);

// Let's do the same thing with `getCode`
bytes memory args = abi.encode(arg1, arg2);
bytes memory bytecode =
abi.encodePacked(vm.getCode("MyContract.sol:MyContract"), args);
address anotherAddress;
assembly {
    anotherAddress := create(0, add(bytecode, 0x20), mload(bytecode))
}

assertEq0(address(myContract).code, anotherAddress.code); // [PASS]
```

Deploy a contract to an arbitrary address by combining `getCode` and `etch`

```
// Deploy
bytes memory args = abi.encode(arg1, arg2);
bytes memory bytecode =
abi.encodePacked(vm.getCode("MyContract.sol:MyContract"), args);
address deployed;
assembly {
    deployed := create(0, add(bytecode, 0x20), mload(bytecode))
}

// Set the bytecode of an arbitrary address
vm.etch(targetAddr, deployed.code);
```

SEE ALSO

`getDeployedCode` `eth`

Forge Standard Library

`deployCode` `deployCodeTo`

getDeployedCode

Signature

```
function getDeployedCode(string calldata) external returns (bytes memory);
```

Description

This cheatcode works similar to `getCode` but only returns the **deployed** bytecode (aka runtime bytecode) for a contract in the project given the path to the contract.

The main use case for this cheat code is as a shortcut to deploy stateless contracts to arbitrary addresses.

The calldata parameter can either be in the form `ContractFile.sol` (if the filename and contract name are the same) , `ContractFile.sol:ContractName` , or the path to an artifact, relative to the root of your project.

ℹ️ Note

`getDeployedCode` requires read permission for the output directory, see file cheatcodes.

To grant read access set `fs_permissions = [{ access = "read", path = "./out"}]` in your `foundry.toml` .

Examples

Deploy a stateless contract at an arbitrary address using `getDeployedCode` and `etch` .

```
// A stateless contract that we want deployed at a specific address
contract Override {
    event Payload(address sender, address target, bytes data);

    function emitPayload(
        address target, bytes calldata message
    ) external payable returns (uint256) {
        emit Payload(msg.sender, target, message);
        return 0;
    }
}

// get the **deployedBytecode**
bytes memory code = vm.getDeployedCode("Override.sol:Override");

// set the code of an arbitrary address
address overrideAddress = address(64);
vm.etch(overrideAddress, code);
assertEq(overrideAddress.code, code);
```

SEE ALSO

Forge Standard Library

`getCode` `etch`

sleep

Signature

```
function sleep(uint256 milliseconds) external;
```

Description

Sleeps for a given amount of milliseconds.

Examples

```
vm.sleep(10_000); // Halts execution for 10 seconds
```

unixTime

Signature

```
function unixTime() external returns (uint milliseconds);
```

Description

Returns the time since unix epoch in milliseconds.

Examples

```
uint start = vm.unixTime();
vm.sleep(10_000); // Halts execution for 10 seconds
uint end = vm.unixTime();
assertEq(end - start, 10_000);
```

setEnv

Signature

```
function setEnv(string calldata key, string calldata value) external;
```

Description

Set an environment variable `key=value`.

> ℹ️ **Note**
>
> Environment variables set by a process are only accessible by itself and its child processes. Thus, calling `setEnv` will only modify environment variables of the currently running `forge` process, and won't affect the shell (`forge` 's parent process), i.e., the they won't persist after the `forge` process exit.

Tips

- The environment variable key can't be empty.
- The environment variable key can't contain the equal sign `=` or the NUL character `\0`.
- The environment variable value can't contain the NUL character `\0`.

Examples

```
string memory key = "hello";
string memory val = "world";
cheats.setEnv(key, val);
```

envOr

Signature

```
function envOr(string calldata key, bool defaultValue) external returns (bool
value);
function envOr(string calldata key, uint256 defaultValue) external returns
(uint256 value);
function envOr(string calldata key, int256 defaultValue) external returns
(int256 value);
function envOr(string calldata key, address defaultValue) external returns
(address value);
function envOr(string calldata key, bytes32 defaultValue) external returns
(bytes32 value);
function envOr(string calldata key, string calldata defaultValue) external
returns (string memory value);
function envOr(string calldata key, bytes calldata defaultValue) external
returns (bytes memory value);
```

```
function envOr(string calldata key, string calldata delimiter, bool[] calldata
defaultValue) external returns (bool[] memory value);
function envOr(string calldata key, string calldata delimiter, uint256[]
calldata defaultValue) external returns (uint256[] memory value);
function envOr(string calldata key, string calldata delimiter, int256[]
calldata defaultValue) external returns (int256[] memory value);
function envOr(string calldata key, string calldata delimiter, address[]
calldata defaultValue) external returns (address[] memory value);
function envOr(string calldata key, string calldata delimiter, bytes32[]
calldata defaultValue) external returns (bytes32[] memory value);
function envOr(string calldata key, string calldata delimiter, string[]
calldata defaultValue) external returns (string[] memory value);
function envOr(string calldata key, string calldata delimiter, bytes[] calldata
defaultValue) external returns (bytes[] memory value);
```

Description

A non-failing way to read an environment variable of any type: if the requested environment key does not exist, `envOr()` will return a default value instead of reverting (works with arrays too).

The returned type is determined by the type of `defaultValue` parameter passed.

Tips

- Use `envOr(key, defaultValue)` to read a single value
- Use `envOr(key, delimiter, defaultValue[])` to read an array with delimiter
- The parsing of the environment variable will be done according to the type of `defaultValue` (e.g. if the default value type is `uint` - the environment variable will be also parsed as `uint`)
- Use explicit casting for literals to specify type of default variable: `uint(69)` will return an `uint` but `int(69)` will return an `int`
- Same with: `string("")` and `bytes("")` - these will return `string` and `bytes` accordingly
- Use dynamic arrays (`bool[]`) instead of fixed-size arrays (`bool[4]`) when providing default values (only dynamic arrays are supported)

Examples

Single Value

If the environment variable `FORK` is not set, you can specify it to be `false` by default:

```
bool fork = vm.envOr("FORK", false);
```

or

```
address owner;

function setUp() {
  owner = vm.envOr("OWNER", address(this));
}
```

Array

If the environment variable `BAD_TOKENS` is not set, you can specify the default to be an empty array:

```
address[] badTokens;

function envBadTokens() public {
  badTokens = vm.envOr("BAD_TOKENS", ",", badTokens);
}
```

or

```
function envBadTokens() public {
  address[] memory defaultBadTokens = new address[](0);
  address[] memory badTokens = vm.envOr("BAD_TOKENS", ",", defaultBadTokens);
}
```

envBool

Signature

```
function envBool(string calldata key) external returns (bool value);
```

```
function envBool(string calldata key, string calldata delimiter) external
returns (bool[] memory values);
```

Description

Read an environment variable as `bool` or `bool[]`.

Tips

- For `true`, use either "true" or "True" for the environment variable value.
- For `false`, use either "false" or "False" for the environment variable value.
- For arrays, you can specify the delimiter used to separate the values with the `delimiter` parameter.

Examples

Single Value

With environment variable `BOOL_VALUE=true`,

```
string memory key = "BOOL_VALUE";
bool expected = true;
bool output = cheats.envBool(key);
assert(output == expected);
```

Array

With environment variable `BOOL_VALUES=true,false,True,False`,

```
string memory key = "BOOL_VALUES";
string memory delimiter = ",";
bool[4] memory expected = [true, false, true, false];
bool[] memory output = cheats.envBool(key, delimiter);
assert(keccak256(abi.encodePacked((output))) ==
keccak256(abi.encodePacked((expected))));
```

envUint

Signature

```
function envUint(string calldata key) external returns (uint256 value);
```

```
function envUint(string calldata key, string calldata delimiter) external
returns (uint256[] memory values);
```

Description

Read an environment variable as `uint256` or `uint256[]`.

Tips

- If the value starts with `0x`, it will be interpreted as a hex value, otherwise, it will be treated as a decimal number.
- For arrays, you can specify the delimiter used to separate the values with the `delimiter` parameter.

Examples

Single Value

With environment variable `UINT_VALUE=115792089237316195423570985008687907853269984665640564039457584007913129639935`,

```
string memory key = "UINT_VALUE";
uint256 expected = type(uint256).max;
uint256 output = cheats.envUint(key);
assert(output == expected);
```

Array

With environment variable `UINT_VALUES=0,0x0000000000000000000000000000000000000000000000000000000000000000`,

```
string memory key = "UINT_VALUES";
string memory delimiter = ",";
uint256[2] memory expected = [type(uint256).min, type(uint256).min];
uint256[] memory output = cheats.envUint(key, delimiter);
assert(keccak256(abi.encodePacked((output))) ==
keccak256(abi.encodePacked((expected))));
```

envInt

Signature

```
function envInt(string calldata key) external returns (int256 value);
```

```
function envInt(string calldata key, string calldata delimiter) external
returns (int256[] memory values);
```

Description

Read an environment variable as `int256` or `int256[]`.

Tips

- If the value starts with `0x`, `-0x` or `+0x`, it will be interpreted as a hex value, otherwise, it will be treated as a decimal number.
- For arrays, you can specify the delimiter used to separate the values with the `delimiter` parameter.

Examples

Single Value

With environment variable `INT_VALUE=-57896044618658097711785492504343953926634992332820282019728792003956564819968`,

```
string memory key = "INT_VALUE";
int256 expected = type(int256).min;
int256 output = cheats.envInt(key);
assert(output == expected);
```

Array

With environment variable `INT_VALUES=-0x8000000000000000000000000000000000000000000000000000000000000000,+0`

```
x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF,
```

```
string memory key = "INT_VALUES";
string memory delimiter = ",";
int256[2] memory expected = [type(int256).min, type(int256).max];
int256[] memory output = cheats.envInt(key, delimiter);
assert(keccak256(abi.encodePacked((output))) ==
keccak256(abi.encodePacked((expected))));
```

envAddress

Signature

```
function envAddress(string calldata key) external returns (address value);
```

```
function envAddress(string calldata key, string calldata delimiter) external
returns (address[] memory values);
```

Description

Read an environment variable as `address` or `address[]`.

Tips

- For arrays, you can specify the delimiter used to separate the values with the `delimiter` parameter.

Examples

Single Value

With environment variable `ADDRESS_VALUE=0x7109709ECfa91a80626fF3989D68f67F5b1DD12D`,

```
string memory key = "ADDRESS_VALUE";
address expected = 0x7109709ECfa91a80626fF3989D68f67F5b1DD12D;
address output = cheats.envAddress(key);
assert(output == expected);
```

Array

With environment variable `ADDRESS_VALUES=0x7109709ECfa91a80626fF3989D68f67F5b1DD12D,0x0000000000000000000000000000000000000000000000000000`,

```
    string memory key = "ADDRESS_VALUES";
    string memory delimiter = ",";
    address[2] memory expected = [
        0x7109709ECfa91a80626fF3989D68f67F5b1DD12D,
        0x0000000000000000000000000000000000000000
    ];
    address[] memory output = cheats.envAddress(key, delimiter);
    assert(keccak256(abi.encodePacked((output))) ==
    keccak256(abi.encodePacked((expected))));
```

envBytes32

Signature

```
function envBytes32(string calldata key) external returns (bytes32 value);
```

```
function envBytes32(string calldata key, string calldata delimiter) external
returns (bytes32[] memory values);
```

Description

Read an environment variable as `bytes32` or `address[]`.

Tips

- For arrays, you can specify the delimiter used to separate the values with the `delimiter` parameter.

Examples

Single Value

With environment variable `BYTES32_VALUE=0x00`,

```
string memory key = "BYTES32_VALUE";
bytes32 expected =
bytes32(0x0000000000000000000000000000000000000000000000000000000000000000);
bytes32 output = cheats.envBytes32(key);
assert(output == expected);
```

Array

With environment variable
`BYTES32_VALUES=0x7109709ECfa91a80626fF3989D68f67F5b1DD12D,0x00`,

```
    string memory key = "BYTES32_VALUES";
    string memory delimiter = ",";
    bytes32[2] memory expected = [

    bytes32(0x7109709ECfa91a80626fF3989D68f67F5b1DD12D000000000000000000000000),
        bytes32(0x0000000000000000000000000000000000000000000000000000000000000000)
    ];
    bytes32[] memory output = cheats.envBytes32(key, delimiter);
    assert(keccak256(abi.encodePacked((output))) ==
    keccak256(abi.encodePacked((expected))));
```

envString

Signature

```
function envString(string calldata key) external returns (string value);
```

```
function envString(string calldata key, string calldata delimiter) external
returns (string[] memory values);
```

Description

Read an environment variable as `string` or `string[]`. In case the environment variable is not defined, Forge will fail with the following error message:

[FAIL. Reason: Failed to get environment variable `FOO` as type `string`: environment variable not found]

Tips

- You can put your environment variables in a `.env` file. Forge will automatically load them when running `forge test`.
- For arrays, you can specify the delimiter used to separate the values with the `delimiter` parameter.
- Choose a delimiter that doesn't appear in the string values, so that they can be correctly separated.

Examples

Single Value

With environment variable `STRING_VALUE=hello, world!`,

```
string memory key = "STRING_VALUE";
string memory expected = "hello, world!";
string memory output = vm.envString(key);
assertEq(output, expected);
```

Array

With environment variable `STRING_VALUES=hello, world!|0x7109709ECfa91a80626fF3989D68f67F5b1DD12D`;

```
string memory key = "STRING_VALUES";
string memory delimiter = "|";
string[2] memory expected = [
    "hello, world!",
    "0x7109709ECfa91a80626fF3989D68f67F5b1DD12D"
];
string[] memory output = vm.envString(key, delimiter);
for (uint i = 0; i < expected.length; ++i) {
    assert(keccak256(abi.encodePacked((output[i]))) ==
keccak256(abi.encodePacked((expected[i]))));
}
```

envBytes

Signature

```
function envBytes(bytes calldata key) external returns (bytes value);
```

```
function envBytes(bytes calldata key, bytes calldata delimiter) external
returns (bytes[] memory values);
```

Description

Read an environment variable as `bytes` or `bytes[]`.

Tips

- For arrays, you can specify the delimiter used to separate the values with the `delimiter` parameter.

Examples

Single Value

With environment variable `BYTES_VALUE=0x7109709ECfa91a80626fF3989D68f67F5b1DD12D;`

```
bytes memory key = "BYTES_VALUE";
bytes expected = hex"7109709ECfa91a80626fF3989D68f67F5b1DD12D";
bytes output = cheats.envBytes(key);
assertEq(output, expected);
```

Array

With environment variable
`BYTES_VALUE=0x7109709ECfa91a80626fF3989D68f67F5b1DD12D,0x00;`

```
bytes memory key = "BYTES_VALUES";
bytes memory delimiter = ",";
bytes[] memory expected = new bytes[](2);
expected[0] = hex"7109709ECfa91a80626fF3989D68f67F5b1DD12D";
expected[1] = hex"00";
bytes[] memory output = cheats.envBytes(key, delimiter);
for (uint i = 0; i < expected.length; ++i) {
    assert(keccak256(abi.encodePacked((output[i]))) ==
keccak256(abi.encodePacked((expected[i]))));
}
```

keyExists

Status

`keyExists` is being deprecated in favor of `keyExistsJson`. It will be removed in future versions.

Signature

```
// Check if a key exists in a JSON string.
vm.keyExists(string memory json, string memory key) returns (bool)
```

Description

Checks if a key exists in a JSON string.

Examples

```
string memory path = "./path/to/jsonfile.json";
string memory json = vm.readFile(path);
bool exists = vm.keyExists(json, ".key");
assertTrue(exists);
```

keyExistsJson

Signature

```
// Check if a key exists in a JSON string.
vm.keyExistsJson(string memory json, string memory key) returns (bool)
```

Description

Checks if a key exists in a JSON string.

Examples

```
string memory path = "./path/to/jsonfile.json";
string memory json = vm.readFile(path);
bool exists = vm.keyExistsJson(json, ".key");
assertTrue(exists);
```

keyExistsToml

Signature

```
// Check if a key exists in a TOML table.
vm.keyExistsToml(string memory toml, string memory key) returns (bool)
```

Description

Checks if a key exists in a TOML table.

Examples

```
string memory path = "./path/to/tomlfile.toml";
string memory toml = vm.readFile(path);
bool exists = vm.keyExistsToml(toml, ".key");
assertTrue(exists);
```

parseJson

Signature

```
// Return the value(s) that correspond to 'key'
vm.parseJson(string memory json, string memory key)
// Return the entire JSON file
vm.parseJson(string memory json);
```

Description

These cheatcodes are used to parse JSON files in the form of strings. Usually, it's coupled with `vm.readFile()` which returns an entire file in the form of a string.

You can use `stdJson` from `forge-std`, as a helper library for better UX.

The cheatcode accepts either a `key` to search for a specific value in the JSON, or no key to return the entire JSON. It returns the value as an abi-encoded `bytes` array. That means that you will have to `abi.decode()` to the appropriate type for it to function properly, else it will `revert`.

JSONpath Key

`parseJson` uses a syntax called JSONpath to form arbitrary keys for arbitrary JSON files. The same syntax (or rather a dialect) is used by the tool `jq`.

To read more about the syntax, you can visit the README of the rust library that we use under the hood to implement the feature. That way you can be certain that you are using the correct dialect of jsonPath.

JSON Encoding Rules

We use the terms `number`, `string`, `object`, `array`, `boolean`, `null` as they are defined in the JSON spec.

Encoding Rules

- `null` is encoded as `bytes32(0)`
- Numbers >= 0 are encoded as `uint256`

- Negative numbers are encoded as `int256`
- Floating point numbers with decimal digitals are not allowed.
- Floating point numbers using the scientific notation can be `uint256` or `int256` depending on the value.
- A string that can be decoded into a type of `H160` and starts with `0x` is encoded as an `address`. In other words, if it can be decoded into an address, it's probably an address
- A string that starts with `0x` is encoded as `bytes32` if it has a length of `66` or else to `bytes`
- A string that is neither an `address`, a `bytes32` or `bytes`, is encoded as a `string`
- An array is encoded as a dynamic array of the type of its first element
- An object (`{}`) is encoded as a `tuple`

Type Coercion

As described above, `parseJson` needs to deduce the type of JSON value and that has some inherent limitations. For that reason, there is a sub-family of `parseJson*` cheatcodes that coerce the type of the returned value.

For example `vm.parseJsonUint(json, key)` will coerce the value to a `uint256`. That means that it can parse all the following values and return them as a `uint256`. That includes a number as type `number`, a stringified number as a `string` and of course it's hex representation.

```
{
  "hexUint": "0x12C980",
  "stringUint":
"115792089237316195423570985008687907853269984665640564039457584007913129639935
",
  "numberUint":
115792089237316195423570985008687907853269984665640564039457584007913129639935
}
```

Similarly, there are cheatcodes for all types (including `bytes` and `bytes32`) and their arrays (`vm.parseJsonUintArray`).

Decoding JSON objects into Solidity structs

JSON objects are encoded as tuples, and can be decoded via tuples or structs. That means that you can define a `struct` in Solidity and it will decode the entire JSON object into that `struct`.

For example:

The following JSON:

```
{
  "a": 43,
  "b": "sigma"
}
```

will be decoded into:

```
struct Json {
    uint256 a;
    string b;
}
```

As the values are returned as an abi-encoded tuple, the exact name of the attributes of the struct don't need to match the names of the keys in the JSON. The above json file could also be decoded as:

```
struct Json {
    uint256 apple;
    string pineapple;
}
```

What matters is the alphabetical order. As the JSON object is an unordered data structure but the tuple is an ordered one, we had to somehow give order to the JSON. The easiest way was to order the keys by alphabetical order. That means that in order to decode the JSON object correctly, you will need to define attributes of the struct with **types** that correspond to the values of the alphabetical order of the keys of the JSON.

- The struct is interpreted serially. That means that the tuple's first item will be decoded based on the first item of the struct definition (no alphabetical order).
- The JSON will parsed alphabetically, not serially.
- Note that this parsing uses Rust's BTreeMap crate under the hood, meaning that uppercase and lowercase strings are treated differently. Uppercase characters *precede* lowercase in this lexicographical ordering, ie "Zebra" would precede "apple".

Thus, the first (in alphabetical order) value of the JSON, will be abi-encoded and then tried to be abi-decoded, based on the type of the first attribute of the `struct`.

The above JSON would not be able to be decoded with the struct below:

```
struct Json {
    uint256 b;
    uint256 a;
}
```

The reason is that it would try to decode the string `"sigma"` as a uint. To be exact, it would be decoded, but it would result to a wrong number, since it would interpret the bytes incorrectly.

Decoding JSON Objects, a tip

If your JSON object has `hex numbers`, they will be encoded as bytes. The way to decode them as `uint` for better UX, is to define two `struct`, one intermediary with the definition of these values as `bytes` and then a final `struct` that will be consumed by the user.

1. Decode the JSON into the intermediary `struct`
2. Convert the intermediary struct to the final one, by converting the `bytes` to `uint`. We have a helper function in `forge-std` to do this
3. Give the final `struct` to the user for consumption

How to use StdJson

1. Import the library `import "../StdJson.sol";`
2. Define its usage with `string`: `using stdJson for string;`
3. If you want to parse simple values (numbers, address, etc.) use the helper functions
4. If you want to parse entire JSON objects:
    1. Define the `struct` in Solidity. Make sure to follow the alphabetical order – it's hard to debug
    2. Use the `parseRaw()` helper function to return abi-encoded `bytes` and then decode them to your struct

```
string memory root = vm.projectRoot();
string memory path = string.concat(root,
"/src/test/fixtures/broadcast.log.json");
string memory json = vm.readFile(path);
bytes memory transactionDetails = json.parseRaw(".transactions[0].tx");
RawTx1559Detail memory rawTxDetail = abi.decode(transactionDetails,
(RawTx1559Detail));
```

Forge script artifacts

We have gone ahead and created a handful of helper struct and functions to read the artifacts from broadcasting a forge script.

Currently, we only support artifacts produced by EIP1559-compatible chains and we **don't** support yet the parsing of the entire `broadcast.json` artifact. You will need to parse for individual values such as the `transactions`, the `receipts`, etc.

To read the transactions, it's as easy as doing:

```
function testReadEIP1559Transactions() public {
    string memory root = vm.projectRoot();
    string memory path = string.concat(root,
"/src/test/fixtures/broadcast.log.json");
    Tx1559[] memory transactions = readTx1559s(path);
}
```

and then you can access their various fields in these structs:

```
struct Tx1559 {
    string[] arguments;
    address contractAddress;
    string contractName;
    string functionSig;
    bytes32 hash;
    Tx1559Detail txDetail;
    string opcode;
}

struct Tx1559Detail {
    AccessList[] accessList;
    bytes data;
    address from;
    uint256 gas;
    uint256 nonce;
    address to;
    uint256 txType;
    uint256 value;
}
```

Troubleshooting

Cannot read file

FAIL. Reason: The path `<file-path>` is not allowed to be accessed for read operations

---

If you receive this error, make sure that you enable read permissions in `foundry.toml` using the `fs_permissions` key

References

- Helper Library: stdJson.sol
- Usage examples: stdCheats.t.sol
- File Cheatcodes: cheatcodes for working with files

parseToml

Signature

```
// Return the value(s) that correspond to 'key'
vm.parseToml(string memory toml, string memory key)
// Return the entire TOML file
vm.parseToml(string memory toml);
```

Description

These cheatcodes are used to parse TOML files in the form of strings after converting to JSON. Usually, it's coupled with `vm.readFile()` which returns an entire file in the form of a string.

You can use `stdToml` from `forge-std`, as a helper library for better UX.

The cheatcode accepts either a `key` to search for a specific value in the TOML, or no key to return the entire TOML. It returns the value as an abi-encoded `bytes` array. That means that you will have to `abi.decode()` to the appropriate type for it to function properly, else it will `revert`.

JSONpath Key

`parseToml` uses a syntax called JSONpath to form arbitrary keys for arbitrary JSON files. The same syntax (or rather a dialect) is used by the tool `jq`.

To read more about the syntax, you can visit the README of the rust library that we use under the hood to implement the feature. That way you can be certain that you are using the correct dialect of jsonPath.

Encoding Rules

We use the terms `string`, `integer`, `float`, `boolean`, `array`, `datetime`, `inline-table` as they are defined in the TOML spec.

We use the terms `number`, `string`, `object`, `array`, `boolean`, `null` as they are defined in the JSON spec.

TOML Encoding Rules

- `float` is limited to 32 bits (i.e. `+1.5`). It is recommended to use strings to prevent precision loss
- `integer` is limited to 64 bits (i.e. `9223372036854775807`). It is recommended to use strings to encode large values
- Array values cannot have mixed types (i.e. `[256, "b"]`, only `[256, 512]` or `["a", "b"]`)
- `datetime` is encoded as a `string` upon conversion
- `float` is encoded as a `number` upon conversion
- `integer` is encoded as a `number` upon conversion
- `inline-table` (or `table`) is encoded as `object` upon conversion
- `null` is encoded as a `"null"` string

JSON Encoding Rules

- `null` is encoded as `bytes32(0)` or `""`
- Numbers >= 0 are encoded as `uint256`
- Negative numbers are encoded as `int256`
- Floating point numbers with decimal digits are not allowed
- Floating point numbers using the scientific notation can be `uint256` or `int256` depending on the value
- A string that can be decoded into a type of `H160` and starts with `0x` is encoded as an `address`. In other words, if it can be decoded into an address, it's probably an address
- A string that starts with `0x` is encoded as `bytes32` if it has a length of `66` or else to `bytes`
- A string that is neither an `address`, a `bytes32` or `bytes`, is encoded as a `string`
- An array is encoded as a dynamic array of the type of its first element
- An object (`{}`) is encoded as a `tuple`

Type Coercion

As described above, `parseToml` needs to deduce the type of TOML value and that has some inherent limitations. For that reason, there is a sub-family of `parseToml*` cheatcodes that coerce the type of the returned value.

For example `vm.parseTomlUint(toml, key)` will coerce the value to a `uint256`. That means that it can parse all the following values and return them as a `uint256`. That includes a number as type `number`, a stringified number as a `string` and of course it's hex representation.

```
hexUint = "0x12C980"
stringUint =
"115792089237316195423570985008687907853269984665640564039457584007913129639935
"
numberUint = 9223372036854775807 # TOML is limited to 64-bit integers
```

Similarly, there are cheatcodes for all types (including `bytes` and `bytes32`) and their arrays (`vm.parseTomlUintArray`).

Decoding TOML tables into Solidity structs

TOML tables are converted to JSON objects. JSON objects are encoded as tuples, and can be decoded via tuples or structs. That means that you can define a `struct` in Solidity and it will decode the entire JSON object into that `struct`.

For example:

The following TOML:

```
a = 43
b = "sigma"
```

will be converted into the following JSON:

```
{
  "a": 43,
  "b": "sigma"
}
```

will be decoded into:

```
struct Json {
    uint256 a;
    string b;
}
```

As the values are returned as an abi-encoded tuple, the exact name of the attributes of the struct don't need to match the names of the keys in the JSON. The above json file could also be decoded as:

```
struct Json {
    uint256 apple;
    string pineapple;
}
```

What matters is the alphabetical order. As the JSON object is an unordered data structure but the tuple is an ordered one, we had to somehow give order to the JSON. The easiest way

was to order the keys by alphabetical order. That means that in order to decode the JSON object correctly, you will need to define attributes of the struct with **types** that correspond to the values of the alphabetical order of the keys of the JSON.

- The struct is interpreted serially. That means that the tuple's first item will be decoded based on the first item of the struct definition (no alphabetical order).
- The JSON will parsed alphabetically, not serially.
- Note that this parsing uses Rust's BTreeMap crate under the hood, meaning that uppercase and lowercase strings are treated differently. Uppercase characters *precede* lowercase in this lexicographical ordering, ie "Zebra" would precede "apple".

Thus, the first (in alphabetical order) value of the JSON, will be abi-encoded and then tried to be abi-decoded, based on the type of the first attribute of the `struct`.

The above TOML would not be able to be decoded with the struct below:

```
struct Json {
    uint256 b;
    uint256 a;
}
```

The reason is that it would try to decode the string `"sigma"` as a uint. To be exact, it would be decoded, but it would result to a wrong number, since it would interpret the bytes incorrectly.

How to use StdToml

1. Import the library `import "../StdToml.sol";`
2. Define its usage with `string`: `using stdToml for string;`
3. If you want to parse simple values (numbers, address, etc.) use the helper functions
4. If you want to parse entire TOML tables:
    1. Define the `struct` in Solidity. Make sure to follow the alphabetical order – it's hard to debug
    2. Use the `parseRaw()` helper function to return abi-encoded `bytes` and then decode them to your struct

```
string memory root = vm.projectRoot();
string memory path = string.concat(root, "/src/test/fixtures/config.toml");
string memory toml = vm.readFile(path);
bytes memory data = toml.parseRaw(".");
Config memory config = abi.decode(data, (Config))
```

Troubleshooting

Cannot read file

FAIL. Reason: The path `<file-path>` is not allowed to be accessed for read operations

---

If you receive this error, make sure that you enable read permissions in `foundry.toml` using the `fs_permissions` key

References

- Helper Library: stdToml.sol
- File Cheatcodes: cheatcodes for working with files

parseJsonKeys

Signature

```
// Get list of keys present in a JSON string
function parseJsonKeys(string calldata json, string calldata key) external pure
returns (string[] memory keys);
```

Description

Gets list of keys present in a JSON string

Examples

```
string memory json = '{"key": {"a": 1, "b": 2}}';
string[] memory keys = vm.parseJsonKeys(json, ".key"); // ["a", "b"]
```

```
string memory json = '{"key": "something"}';
string[] memory keys = vm.parseJsonKeys(json, "$"); // ["key"]
```

```
string memory json = '{"root_key": [{"a": 1, "b": 2}]}';
string[] memory keys = vm.parseJsonKeys(json, ".root_key.0"); // ["a", "b"]
```

parseTomlKeys

Signature

```
// Get list of keys present in a TOML string
function parseTomlKeys(string calldata toml, string calldata key) external pure
returns (string[] memory keys);
```

Description

Gets list of keys present in a TOML string

Examples

```
// [key]
// a = 1
// b = 2

string memory toml = '[key]\n a = 1\n b = 2';
string[] memory keys = vm.parseTomlKeys(toml, ".key"); // ["a", "b"]
```

```
// key = "something"

string memory toml = 'key = \"something\"';
string[] memory keys = vm.parseTomlKeys(toml, "$"); // ["key"]
```

```
// [[root_key]]
// a = 1
// b = 2

string memory toml = '[[root_key]]\n a = 1\n b = 2';
string[] memory keys = vm.parseTomlKeys(toml, ".root_key.0"); // ["a", "b"]
```

serializeJson

Signature

```
function serializeJson(string calldata objectKey, string calldata value)
    external
    returns (string memory json);

function serializeBool(string calldata objectKey, string calldata valueKey,
bool value)
    external
    returns (string memory json);

function serializeUint(string calldata objectKey, string calldata valueKey,
uint256 value)
    external
    returns (string memory json);

function serializeInt(string calldata objectKey, string calldata valueKey,
int256 value)
    external
    returns (string memory json);

function serializeAddress(string calldata objectKey, string calldata valueKey,
address value)
    external
    returns (string memory json);

function serializeBytes32(string calldata objectKey, string calldata valueKey,
bytes32 value)
    external
    returns (string memory json);

function serializeString(string calldata objectKey, string calldata valueKey,
string calldata value)
    external
    returns (string memory json);

function serializeBytes(string calldata objectKey, string calldata valueKey,
bytes calldata value)
    external
    returns (string memory json);

function serializeBool(string calldata objectKey, string calldata valueKey,
bool[] calldata values)
    external
    returns (string memory json);

function serializeUint(string calldata objectKey, string calldata valueKey,
uint256[] calldata values)
    external
    returns (string memory json);

function serializeInt(string calldata objectKey, string calldata valueKey,
```

```
int256[] calldata values)
    external
    returns (string memory json);

function serializeAddress(string calldata objectKey, string calldata valueKey,
address[] calldata values)
    external
    returns (string memory json);

function serializeBytes32(string calldata objectKey, string calldata valueKey,
bytes32[] calldata values)
    external
    returns (string memory json);

function serializeString(string calldata objectKey, string calldata valueKey,
string[] calldata values)
    external
    returns (string memory json);

function serializeBytes(string calldata objectKey, string calldata valueKey,
bytes[] calldata values)
    external
    returns (string memory json);
```

Description

Serializes values as a stringified JSON object.

How it works

The idea is that the user serializes the values of the JSON file and finally writes that object to a file. The user needs to pass:

- A key for the *object* to which the value should be serialized to. This enables the user to serialize multiple objects in parallel
- A key for the *value* which will be its key in the JSON file
- The value to be serialized

An exception to this is the `serializeJson` function, which only receives an `objectKey` and a json string `value`. This allows the user to serialize an existing json object and directly assign it to the provided `objectKey`. If the `objectKey` is already in use, the whole serialized json is overwritten.

The keys do not need to be of some specific form. They are of type `string` to enable for intuitive human interpretation. Semantically, they are not important other than to be used as keys.

The cheatcodes return the JSON object that is being serialized **up to that point**. That way the user can serialize inner JSON objects and then serialize them in bigger JSON objects,

enabling the user to create arbitrary JSON objects.

Finally, the user writes the JSON object to a JSON file by using writeJson. Alternatively, the user can write the JSON object to TOML file by using writeToml.

Remember: The file path needs to be in the allowed paths. Read more in File cheatcodes.

Example

Let's assume we want to write the following JSON to a file:

{ "boolean": true, "number": 342, "object": { "title": "finally json serialization" } }

```
string memory obj1 = "some key";
vm.serializeBool(obj1, "boolean", true);
vm.serializeUint(obj1, "number", uint256(342));

string memory obj2 = "some other key";
string memory output = vm.serializeString(obj2, "title", "finally json
serialization");

// IMPORTANT: This works because `serializeString` first tries to interpret
`output` as
//   a stringified JSON object. If the parsing fails, then it treats it as a
normal
//   string instead.
//   For instance, an `output` equal to '{ "ok": "asd" }' will produce an
object, but
//   an output equal to '"ok": "asd" }' will just produce a normal string.
string memory finalJson = vm.serializeString(obj1, "object", output);

vm.writeJson(finalJson, "./output/example.json");
```

SEE ALSO

- writeJson

writeJson

Signature

```
function writeJson(string calldata json, string calldata path) external;

function writeJson(string calldata json, string calldata path, string calldata
valueKey) external;
```

Description

Writes a serialized JSON object to a file.

The argument `json` must be a JSON object in stringified form. For example:

```
{ "boolean": true, "number": 342, "object": { "title": "finally json
serialization" } }
```

This is usually built through serializeJson.

The argument `path` is the path of the JSON file to write to.

If no `valueKey` is provided, then the JSON object will be written to a new file. If the file already exists, it will be overwritten.

If a `valueKey` is provided, then the file must already exist and be a valid JSON file. The object in that file will be updated by replacing the value at the *JSON path* `valueKey` with the JSON object `json`.

This is useful to replace some values in a JSON file without having to first parse and then reserialize it. Note that the JSON path must indicate an existing key, so it's not possible to add new keys this way.

Remember: The file path `path` needs to be in the allowed paths. Read more in File cheatcodes.

JSON Paths

Let's consider the following JSON object:

```
{
  "boolean": true,
  "number": 342,
  "obj1": {
    "aNumber": 123,
    "obj2": {
      "aNumber": 123,
      "obj3": {
        "veryDeep": 13371337
      }
    }
  }
}
```

The root object is always assumed, so we can refer to one of its children by starting the path with a dot (`.`). For instance, `.boolean` , `.number` , and `.obj1` . We can go as deep as we want: `.obj1.aNumber` , or `.obj1.obj2.aNumber` . We can even search for a key in a subtree: `.obj1..veryDeep` , or just `..veryDeep` since there's no ambiguity.

See the examples to see this in action.

Examples

A simple example

```
string memory jsonObj = '{ "boolean": true, "number": 342, "myObject": {
"title": "finally json serialization" } }';
vm.writeJson(jsonObj, "./output/example.json");

// replaces the value of `myObject` with a new object
string memory newJsonObj = '{ "aNumber": 123, "aString": "asd" }';
vm.writeJson(newJsonObj, "./output/example.json", ".myObject");

// replaces the value of `aString` in the new object
vm.writeJson("my new string", "./output/example.json", ".myObject.aString");

// Here's example.json:
//
// {
//   "boolean": true,
//   "number": 342,
//   "myObject": {
//     "aNumber": 123,
//     "aString": "my new string"
//   }
// }
```

A more complex example

```
string memory jsonObj = '{ "boolean": true, "number": 342, "obj1": { "foo":
"bar" } }';
vm.writeJson(jsonObj, "./output/example2.json");

string memory jsonObj2 = '{ "aNumber": 123, "obj2": {} }';
vm.writeJson(jsonObj2, "./output/example2.json", ".obj1");

string memory jsonObj3 = '{ "aNumber": 123, "obj3": { "veryDeep": 3 } }';
vm.writeJson(jsonObj3, "./output/example2.json", ".obj1.obj2");

// Here's example2.json so far:
//
// {
//   "boolean": true,
//   "number": 342,
//   "obj1": {
//     "aNumber": 123,
//     "obj2": {
//       "aNumber": 123,
//       "obj3": {
//         "veryDeep": 3
//       }
//     }
//   }
// }

// Note that the JSON object is just the value 13371337 in this case.
vm.writeJson("13371337", "./output/example2.json", "..veryDeep");

// Here's the final example2.json:
//
// {
//   "boolean": true,
//   "number": 342,
//   "obj1": {
//     "aNumber": 123,
//     "obj2": {
//       "aNumber": 123,
//       "obj3": {
//         "veryDeep": 13371337
//       }
//     }
//   }
// }
```

SEE ALSO

- serializeJson

writeToml

Signature

```
function writeToml(string calldata json, string calldata path) external;

function writeToml(string calldata json, string calldata path, string calldata
valueKey) external;
```

Description

Writes a serialized JSON object to a TOML file after conversion.

The argument `json` must be a JSON object in stringified form. For example:

```
{ "boolean": true, "number": 342, "object": { "title": "finally json
serialization" } }
```

This is usually built through serializeJson.

The argument `path` is the path of the TOML file to write to.

If no `valueKey` is provided, then the TOML object will be written to a new file. If the file already exists, it will be overwritten.

If a `valueKey` is provided, then the file must already exist and be a valid TOML file. The object in that file will be updated by replacing the value at the *JSON path* `valueKey` with the JSON object `json` after TOML conversion.

This is useful to replace some values in a TOML file without having to first parse and then reserialize it. Note that the TOML path must indicate an existing key, so it's not possible to add new keys this way.

Remember: The file path `path` needs to be in the allowed paths. Read more in File cheatcodes.

JSON Paths

Let's consider the following JSON object:

```
{
  "boolean": true,
  "number": 342,
  "obj1": {
    "aNumber": 123,
    "obj2": {
      "aNumber": 123,
      "obj3": {
        "veryDeep": 13371337
      }
    }
  }
}
```

The root object is always assumed, so we can refer to one of its children by starting the path with a dot (`.`). For instance, `.boolean`, `.number`, and `.obj1`. We can go as deep as we want: `.obj1.aNumber`, or `.obj1.obj2.aNumber`. We can even search for a key in a subtree: `.obj1..veryDeep`, or just `..veryDeep` since there's no ambiguity.

See the examples to see this in action.

Examples

A simple example

```
string memory jsonObj = '{ "boolean": true, "number": 342, "myObject": {
"title": "finally json serialization" } }';
vm.writeToml(jsonObj, "./output/example.toml");

// replaces the value of `myObject` with a new object
string memory newJsonObj = '{ "aNumber": 123, "aString": "asd" }';
vm.writeToml(newJsonObj, "./output/example.toml", ".myObject");

// replaces the value of `aString` in the new object
vm.writeToml("my new string", "./output/example.toml", ".myObject.aString");

// Here's example.toml:
//
// boolean = true
// number = 342

// [myObject]
// aNumber = 123
// aString = "my new string"
```

A more complex example

```
string memory jsonObj = '{ "boolean": true, "number": 342, "obj1": { "foo":
"bar" } }';
vm.writeToml(jsonObj, "./output/example2.toml");

string memory jsonObj2 = '{ "aNumber": 123, "obj2": {} }';
vm.writeToml(jsonObj2, "./output/example2.toml", ".obj1");

string memory jsonObj3 = '{ "aNumber": 123, "obj3": { "veryDeep": 3 } }';
vm.writeToml(jsonObj3, "./output/example2.toml", ".obj1.obj2");

// Here's example2.toml so far:
//
// boolean = true
// number = 342
//
// [obj1]
// aNumber = 123
//
// [obj1.obj2]
// aNumber = 123
//
// [obj1.obj2.obj3]
// veryDeep = 3

// Note that the JSON object is just the value 13371337 in this case.
vm.writeToml("13371337", "./output/example2.toml", "..veryDeep");

// Here's the final example2.toml:
//
// boolean = true
// number = 342
//
// [obj1]
// aNumber = 123
//
// [obj1.obj2]
// aNumber = 123
//
// [obj1.obj2.obj3]
// veryDeep = 13371337
```

SEE ALSO

- serializeJson

Utilities

- `addr`
- `sign`
- `label`
- `getLabel`
- `deriveKey`
- `rememberKey`
- `toString`
- `breakpoint`
- `createWallet`

addr

Signature

```
function addr(uint256 privateKey) external returns (address);
```

Description

Computes the address for a given private key.

Examples

```
address alice = vm.addr(1);
emit log_address(alice); // 0x7e5f4552091a69125d5dfcb7b8c2659029395bdf
```

sign

Signature

```
function sign(uint256 privateKey, bytes32 digest) external returns (uint8 v,
bytes32 r, bytes32 s);
```

```
function sign(Wallet memory wallet, bytes32 digest) external returns (uint8 v,
bytes32 r, bytes32 s);
```

Description

Signs a digest `digest` with private key `privateKey` or Wallet `wallet`, returning `(v, r, s)`.

This is useful for testing functions that take signed data and perform an `ecrecover` to verify the signer.

Examples

```
(address alice, uint256 alicePk) = makeAddrAndKey("alice");
emit log_address(alice);
bytes32 hash = keccak256("Signed by Alice");
(uint8 v, bytes32 r, bytes32 s) = vm.sign(alicePk, hash);
address signer = ecrecover(hash, v, r, s);
assertEq(alice, signer); // [PASS]
```

This is useful for testing functions that require a signature:

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity 0.8.17;

import { ECDSA } from "@openzeppelin/contracts/utils/cryptography/ECDSA.sol";

contract SigningExample is Ownable {
    using ECDSA for bytes32;

    address public systemAddress;

    function setSystemAddress(address _address) external onlyOwner {
        systemAddress = _address;
    }

    function purchase(uint256 _amount, string calldata _nonce, bytes calldata
_signature) external payable {
        require(isValidSignature(
            systemAddress,
            keccak256(abi.encodePacked(msg.sender, _amount, _nonce)),
            _signature
            ), "Invalid Signature"
        );

        // mint tokens
    }

    function isValidSignature(address _systemAddress, bytes32 hash, bytes
memory signature) internal view returns (bool) {
        require(_systemAddress != address(0), "Missing System Address");

        bytes32 signedHash = hash.toEthSignedMessageHash();
        return signedHash.recover(signature) == _systemAddress;
    }

}

contract SigningExampleTest is Test {
    using ECDSA for bytes32;

    SigningExample public signingExample;

    uint256 internal userPrivateKey;
    uint256 internal signerPrivateKey;

    function setUp() public {
        signingExample = new SigningExample();

        userPrivateKey = 0xa11ce;
        signerPrivateKey = 0xabc123;

        address signer = vm.addr(signerPrivateKey);
        signingExample.setSystemAddress(signer);
    }

    function testPurchase() public {
        address user = vm.addr(userPrivateKey);
        address signer = vm.addr(signerPrivateKey);
```

```
        uint256 amount = 2;
        string memory nonce = 'QSfd8gQE4WYzO29';

        vm.startPrank(signer);
        bytes32 digest = keccak256(abi.encodePacked(user, amount,
nonce)).toEthSignedMessageHash();
        (uint8 v, bytes32 r, bytes32 s) = vm.sign(signerPrivateKey, digest);
        bytes memory signature = abi.encodePacked(r, s, v); // note the order
here is different from line above.
        vm.stopPrank();

        vm.startPrank(user);
        // Give the user some ETH, just for good measure
        vm.deal(user, 1 ether);

        signingExample.purchase(
            amount,
            nonce,
            signature
        );
        vm.stopPrank();
    }
}
```

Wallet

The Wallet overload is a simple wrapper that uses the wallet's private key to sign the digest

```
Wallet memory alice = vm.createWallet("alice");
bytes32 hash = keccak256("Signed by Alice");
(uint8 v, bytes32 r, bytes32 s) = vm.sign(alice, hash);
address signer = ecrecover(hash, v, r, s);
assertEq(alice.addr, signer); // [PASS]
```

`skip`

Signature

```
function skip(bool skip) external;
```

Description

Marks a test as skipped, conditionally. It must be called at the top of the test to ensure it is skipped without any execution.

If `skip` is called with a false boolean, it will not skip the test.

Tests marked as skipped will appear with a `[SKIPPED]` label on the test runner and on the summary, to easily identify skipped tests.

Examples

```
function testSkip() public {
    vm.skip(true);
    /// This revert will not be reached as this test will be skipped.
    revert("Should not reach this revert");
}

function testNotSkip() public {
    vm.skip(false);
    /// This revert will be reached as this test will not be skipped, and the
test will fail.
    revert("Should reach this revert");
}
```

`label`

Signature

```
function label(address addr, string calldata label) external;
```

Description

Sets a label `label` for `addr` in test traces.

If an address is labelled, the label will show up in test traces instead of the address.

getLabel

Signature

```
function getLabel(address) external returns (string memory);
```

Description

Retrieves the label for an address if it was previously labeled. If not, it returns the address prefixed with `unlabeled:`.

deriveKey

Signature

```
function deriveKey(
  string calldata mnemonic,
  uint32 index
) external returns (uint256);
```

```
function deriveKey(
  string calldata mnemonic,
  string calldata path,
  uint32 index
) external returns (uint256);
```

Description

Derive a private key from a given mnemonic or mnemonic file path.

The first signature derives at the derivation path `m/44'/60'/0'/0/{index}`. The second signature allows you to specify the derivation path as the second parameter.

Examples

Derive the private key from the test mnemonic at path `m/44'/60'/0'/0/0`:

```
string memory mnemonic = "test test test test test test test test test test
test junk";
uint256 privateKey = vm.deriveKey(mnemonic, 0);
```

Derive the private key from the test mnemonic at path `m/44'/60'/0'/1/0`:

```
string memory mnemonic = "test test test test test test test test test test
test junk";
uint256 privateKey = vm.deriveKey(mnemonic, "m/44'/60'/0'/1/", 0);
```

SEE ALSO

- rememberKey

Forge Standard Library:

- deriveRememberKey

parseBytes

Signature

```
function parseBytes(string calldata stringifiedValue) external pure returns
(bytes memory parsedValue);
```

Description

Parses the value of `string` into `bytes`

Examples

```
string memory bytesAsString = "0x00000000000000000000000000000000";
bytes memory stringToBytes = vm.parseBytes(bytesAsString); //
0x00000000000000000000000000000000
```

parseAddress

Signature

```
function parseAddress(string calldata stringifiedValue) external pure returns
(address parsedValue);
```

Description

Parses the value of `string` into `address`

Examples

```
string memory addressAsString = "0x0000000000000000000000000000000000000000";
address stringToAddress = vm.parseAddress(addressAsString); //
0x0000000000000000000000000000000000000000
```

parseUint

Signature

```
function parseUint(string calldata stringifiedValue) external pure returns
(uint256 parsedValue);
```

Description

Parses the value of `string` into `uint256`

Examples

```
string memory uintAsString = "12345";
uint256 stringToUint = vm.parseUint(uintAsString); // 12345
```

parseInt

Signature

```
function parseInt(string calldata stringifiedValue) external pure returns
(int256 parsedValue);
```

Description

Parses the value of `string` into `int256`

Examples

```
string memory intAsString = "-12345";
int256 stringToInt = vm.parseInt(intAsString); // -12345
```

parseBytes32

Signature

```
function parseBytes32(string calldata stringifiedValue) external pure returns
(bytes32 parsedValue);
```

Description

Parses the value of `string` into `bytes32`

Examples

```
string memory bytes32AsString =
"0x0000000000000000000000000000000000000000000000000000000000000000";
bytes32 stringToBytes32 = vm.parseBytes32(bytes32AsString); //
0x0000000000000000000000000000000000000000000000000000000000000000
```

parseBool

Signature

```
function parseBool(string calldata stringifiedValue) external pure returns
(bool parsedValue);
```

Description

Parses the value of `string` into `bool`

Examples

```
string memory boolAsString = "false";
bool stringToBool = vm.parseBool(boolAsString); // false
```

rememberKey

Signature

```
function rememberKey(uint256 privateKey) external returns (address);
```

Description

Stores a private key in forge's local wallet and returns the corresponding address which can later be used for broadcasting.

Examples

Derive the private key from the test mnemonic at path `m/44'/60'/0'/0/0`, remember it in forge's wallet and use it to start broadcasting transactions:

```
string memory mnemonic = "test test test test test test test test test test
test junk";
uint256 privateKey = vm.deriveKey(mnemonic, 0);
address deployer = vm.rememberKey(privateKey);

vm.startBroadcast(deployer);
...
vm.stopBroadcast();
```

Load a private key from the `PRIVATE_KEY` environment variable and use it to start broadcasting transactions:

```
address deployer = vm.rememberKey(vm.envUint("PRIVATE_KEY"));

vm.startBroadcast(deployer);
...
vm.stopBroadcast();
```

SEE ALSO

- deriveKey

Forge Standard Library:

- deriveRememberKey

toString

Signature

```
function toString(address) external returns (string memory);
function toString(bool) external returns (string memory);
function toString(uint256) external returns (string memory);
function toString(int256) external returns (string memory);
function toString(bytes32) external returns (string memory);
function toString(bytes) external returns (string memory);
```

Description

Convert any type to its string version. Very useful for operations that demand strings, such as the cheatcode `ffi`.

Bytes are converted to a string of their hex representation with `0x` at the start, signifying that they are encoded in hex.

Examples

```
uint256 number = 420;
string memory stringNumber = vm.toString(number);
vm.assertEq(stringNumber, "420");
```

```
bytes memory testBytes = hex"7109709ECfa91a80626fF3989D68f67F5b1DD12D";
string memory stringBytes = cheats.toString(testBytes);
assertEq("0x7109709ecfa91a80626ff3989d68f67f5b1dd12d", stringBytes);
```

```
address testAddress =  0x7109709ECfa91a80626fF3989D68f67F5b1DD12D;
string memory stringAddress = cheats.toString(testAddress);
assertEq("0x7109709ECfa91a80626fF3989D68f67F5b1DD12D", stringAddress);
```

breakpoint

Signature

```
function breakpoint(string) external;
function breakpoint(string, bool) external;
```

Description

Places a breakpoint to jump to in the debugger view.

Calling `vm.breakpoint('<char>, true)` is equivalent to `vm.breakpoint('<char>)`, but calling `vm.breakpoint('<char, false)` will erase the breakpoint at `'<char`.

If the char is overwritten, only the last one that was visited in the execution steps is considered.

Examples

```
function testBreakpoint() public {
    vm.breakpoint("a");
}
```

Opening up the debugger in a test environment and pressing `'a` will then place the debugger step at the place where the breakpoint cheatcode was called.

```
┌Address: 0x7fa9385be102ac3eac297483dd6233d62b3e1496 | PC: 1904 | Gas used in call: 2963──────────────────────────────────────────
│0768| JUMPI
│076d| JUMPDEST
│076e| POP
│076f| GAS
│0770| ►CALL
│END CALL
└──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
┌Stack: 12──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
│00| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 7f ff ff ff ff ff a2 24
│01| 00 00 00 00 00 00 00 00 00 00 00 00 71 09 70 9e cf a9 1a 80 62 6f f3 98 9d 68 f6 7f 5b 1d d1 2d
│02| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
│03| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 80
│04| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 64
│05| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 80
└──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
┌Memory (max expansion: 256 bytes)──────────────────────────────────────────────────────────────────────────────────────────────
│000| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
│020| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
│040| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 80
│060| 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
│080| f0 25 9e 92 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
│0a0| 00 00 00 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
└──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────

┌Contract call──────────────────────────────────────────────────────────────────────────────────────────────────────────────────
│ 0  // SPDX-License-Identifier: UNLICENSED
│ 1  pragma solidity ^0.8.13;
│ 2
│ 3  import "forge-std/Test.sol";
│ 4
│ 5  contract Breakpoints is Test {
│ 6      function testBreakpoint() public {
│ 7          vm.breakpoint("a");
│ 8      }
│ 9  }
```

SEE ALSO

debugger

`createWallet`

Signature

```
struct Wallet {
    address addr;
    uint256 publicKeyX;
    uint256 publicKeyY;
    uint256 privateKey;
}
```

```
function createWallet(string calldata) external returns (Wallet memory);
```

```
function createWallet(uint256) external returns (Wallet memory);
```

```
function createWallet(uint256, string calldata) external returns (Wallet
memory);
```

Description

Creates a new Wallet struct when given a parameter to derive the private key from.

Tips

`sign()` and `getNonce()` both have supported function overloads for the Wallet struct as well.

Examples

uint256

```
Wallet memory wallet = vm.createWallet(uint256(keccak256(bytes("1"))));

emit log_uint(wallet.privateKey); // uint256(keccak256(bytes("1")))

emit log_address(wallet.addr); // vm.addr(wallet.privateKey)

emit log_address(
    address(
        uint160(
            uint256(
                keccak256(abi.encode(wallet.publicKeyX, wallet.publicKeyY))
            )
        )
    )
); // wallet.addr

emit log_string(vm.getLabel(wallet.addr)); // ""
```

string

```
Wallet memory wallet = vm.createWallet("bob's wallet");

emit log_uint(wallet.privateKey); // uint256(keccak256(bytes("bob's wallet")))

emit log_address(wallet.addr); // vm.addr(wallet.privateKey)

emit log_address(
    address(
        uint160(
            uint256(
                keccak256(abi.encode(wallet.publicKeyX, wallet.publicKeyY))
            )
        )
    )
); // wallet.addr

emit log_string(vm.getLabel(wallet.addr)); // "bob's wallet"
```

uint256 and string

```
Wallet memory wallet = vm.createWallet(uint256(keccak256(bytes("1"))), "bob's
wallet");

emit log_uint(wallet.privateKey); // uint256(keccak256(bytes("1")))

emit log_address(wallet.addr); // vm.addr(wallet.privateKey)

emit log_address(
    address(
        uint160(
            uint256(
                keccak256(abi.encode(wallet.publicKeyX, wallet.publicKeyY))
            )
        )
    )
); // wallet.addr

emit log_string(vm.getLabel(wallet.addr)); // "bob's wallet"
```

snapshot cheatcodes

Signature

```
// Snapshot the current state of the evm.
// Returns the id of the snapshot that was created.
// To revert a snapshot use `revertTo`
function snapshot() external returns(uint256);
// Revert the state of the evm to a previous snapshot
// Takes the snapshot id to revert to.
// This deletes the snapshot and all snapshots taken after the given snapshot
id.
function revertTo(uint256) external returns(bool);
```

Description

`snapshot` takes a snapshot of the state of the blockchain and returns the identifier of the created snapshot

`revertTo` reverts the state of the blockchain to the given snapshot. This deletes the given snapshot, as well as any snapshots taken after (e.g.: reverting to id 2 will delete snapshots with ids 2, 3, 4, etc.)

Examples

```
struct Storage {
    uint slot0;
    uint slot1;
}

contract SnapshotTest is Test {
    Storage store;
    uint256 timestamp;

    function setUp() public {
        store.slot0 = 10;
        store.slot1 = 20;
        vm.deal(address(this), 5 ether);        // balance = 5 ether
        timestamp = block.timestamp;
    }

    function testSnapshot() public {
        uint256 snapshot = vm.snapshot();       // saves the state

        // let's change the state
        store.slot0 = 300;
        store.slot1 = 400;
        vm.deal(address(this), 500 ether);
        vm.warp(12345);                         // block.timestamp = 12345

        assertEq(store.slot0, 300);
        assertEq(store.slot1, 400);
        assertEq(address(this).balance, 500 ether);
        assertEq(block.timestamp, 12345);

        vm.revertTo(snapshot);                  // restores the state

        assertEq(store.slot0, 10, "snapshot revert for slot 0 unsuccessful");
        assertEq(store.slot1, 20, "snapshot revert for slot 1 unsuccessful");
        assertEq(address(this).balance, 5 ether, "snapshot revert for balance
unsuccessful");
        assertEq(block.timestamp, timestamp, "snapshot revert for timestamp
unsuccessful");
    }
}
```

RPC related cheatcodes

Signature

```
// Returns the URL for a configured alias
function rpcUrl(string calldata alias) external returns (string memory);
// Returns all configured (alias, URL) pairs
function rpcUrls() external returns(string[2][] memory);
/// Performs an Ethereum JSON-RPC request to the current fork URL.
function rpc(string calldata method, string calldata params) external returns
(bytes memory data);
```

Description

Provides cheatcodes to access all RPC endpoints configured in the `rpc_endpoints` object of the `foundry.toml`, and the ability to make `rpc` calls using the configured fork URL.

Examples

The following `rpc_endpoints` in `foundry.toml` registers two RPC aliases:

- `optimism` references the URL directly
- `mainnet` references the `RPC_MAINNET` environment value that is expected to contain the actual URL

Env variables need to be wrapped in `${}`

```
# --snip--
[rpc_endpoints]
optimism = "https://optimism.alchemyapi.io/v2/..."
mainnet = "${RPC_MAINNET}"
```

```
string memory url = vm.rpcUrl("optimism");
assertEq(url, "https://optimism.alchemyapi.io/v2/...");
```

If a ENV var is missing, `rpcUrl()` will revert:

```
vm.expectRevert("Failed to resolve env var `${RPC_MAINNET}` in `RPC_MAINNET`:
environment variable not found");
string memory url = vm.rpcUrl("mainnet");
```

Retrieve all available alias -> URL pairs

```
string[2][] memory allUrls = vm.rpcUrls();
assertEq(allUrls.length, 2);

string[2] memory val = allUrls[0];
assertEq(val[0], "optimism");

string[2] memory env = allUrls[1];
assertEq(env[0], "mainnet");
```

Make an RPC call to `eth_getBalance`

```
// balance at block <https://etherscan.io/block/18332681>
bytes memory result = vm.rpc("eth_getBalance", "
[\"0x8D97689C9818892B700e27F316cc3E41e17fBeb9\", \"0x117BC09\"]")
assertEq(hex"10b7c11bcb51e6", result);
```

SEE ALSO

Forge Config

Config Reference

File cheat codes

Signature

```
// Reads the entire content of file to string, (path) => (data)
function readFile(string calldata) external returns (string memory);
/// Reads the entire content of file as binary. `path` is relative to the
project root.
function readFileBinary(string calldata path) external view returns (bytes
memory data);
/// Reads the directory at the given path recursively, up to `maxDepth`.
/// `maxDepth` defaults to 1, meaning only the direct children of the given
directory will be returned.
/// Follows symbolic links if `followLinks` is true.
function readDir(string calldata path) external view returns (DirEntry[] memory
entries);
// Reads next line of file to string, (path) => (line)
function readLine(string calldata) external returns (string memory);
/// Reads a symbolic link, returning the path that the link points to.
/// This cheatcode will revert in the following situations, but is not limited
to just these cases:
/// - `path` is not a symbolic link.
/// - `path` does not exist.
function readLink(string calldata linkPath) external view returns (string
memory targetPath);
// Writes data to file, creating a file if it does not exist, and entirely
replacing its contents if it does.
// (path, data) => ()
function writeFile(string calldata, string calldata) external;
// Writes line to file, creating a file if it does not exist.
// (path, data) => ()
function writeLine(string calldata, string calldata) external;
// Closes file for reading, resetting the offset and allowing to read it from
beginning with readLine.
// (path) => ()
function closeFile(string calldata) external;
// Removes file. This cheatcode will revert in the following situations, but is
not limited to just these cases:
// - Path points to a directory.
// - The file doesn't exist.
// - The user lacks permissions to remove the file.
// (path) => ()
function removeFile(string calldata) external;
// Returns true if the given path points to an existing entity, else returns
false
// (path) => (bool)
function exists(string calldata) external returns (bool);
// Returns true if the path exists on disk and is pointing at a regular file,
else returns false
// (path) => (bool)
function isFile(string calldata) external returns (bool);
// Returns true if the path exists on disk and is pointing at a directory, else
returns false
```

```
// (path) => (bool)
function isDir(string calldata) external returns (bool);
```

Description

These cheatcodes provided by forge-std can be used for filesystem manipulation operations.

By default, filesystem access is disallowed and requires the `fs_permission` setting in `foundry.toml`:

```
# Configures permissions for cheatcodes that touch the filesystem like
`vm.writeFile`
# `access` restricts how the `path` can be accessed via cheatcodes
#    `read-write` | `true`   => `read` + `write` access allowed (`vm.readFile`
+ `vm.writeFile`)
#    `none`| `false` => no access
#    `read` => only read access (`vm.readFile`)
#    `write` => only write access (`vm.writeFile`)
# The `allowed_paths` further lists the paths that are considered, e.g. `./`
represents the project root directory
# By default _no_ fs access permission is granted, and _no_ paths are allowed
# following example enables read access for the project dir _only_:
#       `fs_permissions = [{ access = "read", path = "./"}]`
fs_permissions = [] # default: all file cheat codes are disabled
```

Examples

Append a line to a file, this will create the file if it does not exist yet

This requires read access to the file / project root

```
fs_permissions = [{ access = "read", path = "./"}]
```

```
string memory path = "output.txt";

string memory line1 = "first line";
vm.writeLine(path, line1);

string memory line2 = "second line";
vm.writeLine(path, line2);
```

Write to and read from a file

This requires read-write access to file / project root:

```
fs_permissions = [{ access = "read-write", path = "./"}]
```

```
string memory path = "file.txt";
string memory data = "hello world";
vm.writeFile(path, data);

assertEq(vm.readFile(path), data);
```

Remove a file

This requires write access to file / project root:

```
fs_permissions = [{ access = "write", path = "./"}]
```

```
string memory path = "file.txt";
vm.removeFile(path);

assertFalse(vm.exists(validPath));
```

Verify that a filesystem path is valid

```
// Verify that path 'foo/files/bar.txt' exists
string memory validPath = "foo/files/bar.txt";
assertTrue(vm.exists(validPath));
```

Verify that a filesystem path points to a file or directory

```
// Verify that path 'foo/file/bar.txt' points to a file
string memory validFilePath = "foo/files/bar.txt";
assertTrue(vm.isFile(validFilePath));

// Verify that 'foo/file' points to a directory
string memory validDirPath = "foo/files";
assertTrue(vm.isDir(validDirPath));
```

Forge Standard Library Reference

Forge Standard Library (Forge Std for short) is a collection of helpful contracts that make writing tests easier, faster, and more user-friendly.

Using Forge Std is the preferred way of writing tests with Foundry.

What's included:

- `Vm.sol` : Up-to-date cheatcodes interface

  ```
  import "forge-std/Vm.sol";
  ```

- `console.sol` and `console2.sol` : Hardhat-style logging functionality

  ```
  import "forge-std/console.sol";
  ```

  Note: `console2.sol` contains patches to `console.sol` that allow Forge to decode traces for calls to the console, but it is not compatible with Hardhat.

  ```
  import "forge-std/console2.sol";
  ```

- `Script.sol` : Basic utilities for Solidity scripting

  ```
  import "forge-std/Script.sol";
  ```

- `Test.sol` : The complete Forge Std experience (more details below)

  ```
  import "forge-std/Test.sol";
  ```

Forge Std's `Test`

The `Test` contract in `Test.sol` provides all the essential functionality you need to get started writing tests.

Simply import `Test.sol` and inherit from `Test` in your test contract:

```
import "forge-std/Test.sol";

contract ContractTest is Test { ...
```

What's included:

- Std Libraries
  - Std Logs: Expand upon the logging events from the DSTest library.
  - Std Assertions: Expand upon the assertion functions from the DSTest library.
  - Std Cheats: Wrappers around Forge cheatcodes for improved safety and DX.
  - Std Errors: Wrappers around common internal Solidity errors and reverts.
  - Std Storage: Utilities for storage manipulation.
  - Std Math: Useful mathematical functions.
  - Script Utils: Utility functions which can be accessed in tests and scripts.
  - Console Logging: Console logging functions.
- A cheatcodes instance `vm`, from which you invoke Forge cheatcodes (see Cheatcodes Reference)

```
vm.startPrank(alice);
```

- All Hardhat `console` functions for logging (see Console Logging)

```
console.log(alice.balance); // or `console2`
```

- All Dappsys Test functions for asserting and logging (see Dappsys Test reference)

```
assertEq(dai.balanceOf(alice), 10000e18);
```

- Utility functions also included in `Script.sol` (see Script Utils)

```
// Compute the address a contract will be deployed at for a given deployer
address and nonce
address futureContract = computeCreateAddress(alice, 1);
```

Std Logs

Std Logs expand upon the logging events from the `DSTest` library.

Events

```
event log_array(uint256[] val);
event log_array(int256[] val);
event log_named_array(string key, uint256[] val);
event log_named_array(string key, int256[] val);
```

Usage

This section provides usage examples.

log_array

```
event log_array(<type>[] val);
```

Where `<type>` can be `int256`, `uint256`, `address`.

Example

```
// Assuming storage
// uint256[] data = [10, 20, 30, 40, 50];

emit log_array(data);
```

log_named_array

```
event log_named_array(string key, <type>[] val);
```

Where `<type>` can be `int256`, `uint256`, `address`.

Example

```
// Assuming storage
// uint256[] data = [10, 20, 30, 40, 50];

emit log_named_array("Data", data);
```

Std Assertions

- `fail`
- `assertFalse`
- `assertEq`
- `assertApproxEqAbs`
- `assertApproxEqRel`

fail

Signature

```
function fail(string memory err) internal virtual;
```

Description

Fail a test with a message if a certain branch or execution point is hit.

Examples

```
function test() external {
    for(uint256 place; place < 10; ++i){
        if(game.leaderboard(place) == address(this)) return;
    }
    fail("Not in the top 10.");
}
```

assertFalse

Signature

```
function assertFalse(bool data) internal virtual;
```

```
function assertFalse(bool data, string memory err) internal virtual;
```

Description

Asserts the `condition` is false.

Examples

```
bool failure = contract.fun();
assertFalse(failure);
```

assertEq

Signature

```
function assertEq(bool a, bool b) internal;

function assertEq(bool a, bool b, string memory err) internal;

function assertEq(bytes memory a, bytes memory b) internal;

function assertEq(bytes memory a, bytes memory b, string memory err) internal;

function assertEq(uint256[] memory a, uint256[] memory b) internal;

function assertEq(int256[] memory a, int256[] memory b) internal;

function assertEq(uint256[] memory a, uint256[] memory b, string memory err)
internal;

function assertEq(int256[] memory a, int256[] memory b, string memory err)
internal;

// legacy helper for asserting two uints shorter than 256 bits:
`assertEqUint(uint8(1), uint128(1));`
function assertEqUint(uint256 a, uint256 b) internal;
```

Description

Asserts `a` is equal to `b`.

Works with `bool`, `bytes`, and `int256` and `uint256` arrays.

assertApproxEqAbs

Signature

```
function assertApproxEqAbs(uint256 a, uint256 b, uint256 maxDelta) internal
virtual;
```

```
function assertApproxEqAbs(uint256 a, uint256 b, uint256 maxDelta, string
memory err) internal virtual;
```

Description

Asserts `a` is approximately equal to `b` with delta in absolute value.

Examples

```
function testFail () external {
    uint256 a = 100;
    uint256 b = 200;

    assertApproxEqAbs(a, b, 90);
}
```

```
[PASS] testFail() (gas: 23169)
Logs:
  Error: a ~= b not satisfied [uint]
    Expected: 200
      Actual: 100
   Max Delta: 90
       Delta: 100
```

assertApproxEqRel

Signature

```
function assertApproxEqRel(uint256 a, uint256 b, uint256 maxPercentDelta)
internal virtual;
```

```
function assertApproxEqRel(uint256 a, uint256 b, uint256 maxPercentDelta,
string memory err) internal virtual;
```

Description

Asserts `a` is approximately equal to `b` with delta in percentage, where `1e18` is 100%.

Examples

```
function testFail () external {
    uint256 a = 100;
    uint256 b = 200;
    assertApproxEqRel(a, b, 0.4e18);
}
```

```
[PASS] testFail() (gas: 23884)
Logs:
  Error: a ~= b not satisfied [uint]
      Expected: 200
        Actual: 100
   Max % Delta: 0.400000000000000000
       % Delta: 0.500000000000000000
```

Std Cheats

- `skip`
- `rewind`
- `hoax`
- `startHoax`
- `deal`
- `deployCode`
- `deployCodeTo`
- `bound`
- `changePrank`
- `makeAddr`
- `makeAddrAndKey`
- `noGasMetering`

skip

Signature

```
function skip(uint256 time) public;
```

Description

Skips forward `block.timestamp` by the specified number of seconds.

Examples

```
assertEq(block.timestamp, 0);
skip(3600);
assertEq(block.timestamp, 3600);
```

rewind

Signature

```
function rewind(uint256 time) public;
```

Description

Rewinds `block.timestamp` by the specified number of seconds.

Examples

```
assertEq(block.timestamp, 3600);
rewind(3600);
assertEq(block.timestamp, 0);
```

hoax

Signature

```
function hoax(address who) public;

function hoax(address who, uint256 give) public;

function hoax(address who, address origin) public;

function hoax(address who, address origin, uint256 give) public;
```

Description

Sets up a `prank` from an address that has some ether.

If the balance is not specified, it will be set to 2^128 wei.

startHoax

Signature

```
function startHoax(address who) public;

function startHoax(address who, uint256 give) public;

function startHoax(address who, address origin) public;

function startHoax(address who, address origin, uint256 give) public;
```

Description

Start a perpetual `prank` from an address that has some ether.

If the balance is not specified, it will be set to 2^128 wei.

deal

Signature

```
function deal(address to, uint256 give) public;

function deal(address token, address to, uint256 give) public;

function deal(address token, address to, uint256 give, bool adjust) public;
```

Description

A wrapper around the `deal` cheatcode that also works for most ERC-20 tokens.

If the alternative signature of `deal` is used, adjusts the token's total supply after setting the balance.

Examples

```
deal(address(dai), alice, 10000e18);
assertEq(dai.balanceOf(alice), 10000e18);
```

deployCode

Signature

```
function deployCode(string memory what) public returns (address);
```

```
function deployCode(string memory what, bytes memory args) public returns
(address);
```

```
function deployCode(string memory what, uint256 val) public returns (address);
```

```
function deployCode(string memory what, bytes memory args, uint256 val) public
returns (address);
```

Description

Deploys a contract by fetching the contract bytecode from the artifacts directory.

The calldata parameter can either be in the form `ContractFile.sol` (if the filename and contract name are the same), `ContractFile.sol:ContractName`, or the path to an artifact, relative to the root of your project.

Values can also be passed by using the `val` parameter. This is necessary if you need to send ETH on construction.

Examples

```
address deployment = deployCode("MyContract.sol", abi.encode(arg1, arg2));
```

SEE ALSO

Forge Standard Library

- `deployCodeTo`

deployCodeTo

Signature

```
function deployCodeTo(string memory what, address where) internal virtual;
```

```
function deployCodeTo(string memory what, bytes memory args, address where)
internal virtual;
```

```
function deployCodeTo(string memory what, bytes memory args, uint256 value,
address where) internal virtual;
```

Description

Pseudo-deploys a contract to an arbitrary address by fetching the contract bytecode from the artifacts directory. This can be used to recreate the production environment.

The calldata parameter can either be in the form `ContractFile.sol` (if the filename and contract name are the same), `ContractFile.sol:ContractName`, or the path to an artifact, relative to the root of your project.

Values can also be passed by using the `value` parameter. This is necessary if you need to send ETH on construction.

Examples

```
deployCodeTo("MyContract.sol", abi.encode(arg1, arg2), arbitraryAddr);
```

SEE ALSO

Forge Standard Library

- `deployCode`

bound

Signature

```
function bound(uint256 x, uint256 min, uint256 max) public returns (uint256
result);
```

Description

A mathematical function for wrapping inputs of fuzz tests into a certain range.

You can use it instead of the `assume` cheatcode to get better performance in some cases. Read more here.

Examples

```
input = bound(input, 99, 101);
```

Returns `99` for input `0`.
Returns `100` for input `1`.
Returns `101` for input `2`.
Returns `99` for input `3`.
And so on.

changePrank

Status

`changePrank` is deprecated.

Signature

```
function changePrank(address who) internal;
```

Description

Stops the active prank with `stopPrank` and passes address to `startPrank`.

Useful for starting a global prank in the `setUp` function and deactivating it in certain tests.

makeAddr

Signature

```
function makeAddr(string memory name) internal returns(address addr);
```

Description

Creates an address derived from the provided `name`.

A `label` is created for the derived address with the provided `name` used as the label value.

Examples

```
address alice = makeAddr("alice");
emit log_address(alice); // 0x328809bc894f92807417d2dad6b7c998c1afdac6
```

makeAddrAndKey

Signature

```
function makeAddrAndKey(string memory name) internal returns(address addr,
uint256 privateKey);
```

Description

Creates an address and private key derived from the provided `name`.

A `label` is created for the derived address with the provided `name` used as the label value.

Examples

```
(address alice, uint256 key) = makeAddrAndKey("alice");
emit log_address(alice); // 0x328809bc894f92807417d2dad6b7c998c1afdac6
emit log_uint(key); //
70564938991660933374592024341600875602376452319261984317470407481576058979585
```

noGasMetering

Signature

```
modifier noGasMetering();
```

Description

A function modifier that turns off gas metering for the life of the function.

Note, there is some gas associated with calling the cheatcode, so you will see some gas usage (albeit small) when using this modifier.

Examples

```
function addInLoop() internal returns (uint256) {
    uint256 b;
    for (uint256 i; i < 10000; i++) {
        b + i;
    }
    return b;
}

function addInLoopNoGas() internal noGasMetering returns (uint256) {
    return addInLoop();
}

function testFunc() external {
  uint256 gas_start = gasleft();
  addInLoop();
  uint256 gas_used = gas_start - gasleft();

  uint256 gas_start_no_metering = gasleft();
  addInLoopNoGas();
  uint256 gas_used_no_metering = gas_start_no_metering - gasleft();

  emit log_named_uint("Gas Metering", gas_used);
  emit log_named_uint("No Gas Metering", gas_used_no_metering);
}
```

```
[PASS] testFunc() (gas: 1887191)
Logs:
  Gas Metering: 1880082
  No Gas Metering: 3024
```

assumeNotPrecompile

Signature

```
function assumeNotPrecompile(address addr) public;

function assumeNotPrecompile(address addr, uint256 chainid) public;
```

Description

Uses `assume` to filter precompile addresses from the fuzz tests.

Optionally, a `chainid` may be specified to filter known precompiles on the respective chain.

SEE ALSO

Precompile Registry

assumePayable

Signature

```
function assumePayable(address addr) public;
```

Description

Uses `assume` to filter addresses that reject Ether transfers.

This makes an external call to the specified `addr` with a no calldata and checks `assume` against the success of the call.

Std Errors

- `assertionError`
- `arithmeticError`
- `divisionError`
- `enumConversionError`
- `encodeStorageError`
- `popError`
- `indexOOBError`
- `memOverflowError`
- `zeroVarError`

`assertionError`

Signature

```
stdError.assertionError
```

Description

The internal Solidity error when an `assert` fails.

arithmeticError

Signature

```
stdError.arithmeticError
```

Description

The internal Solidity error when an arithmetic operation fails, e.g. underflow and overflow.

Example

Assume we have a basic vault contract that can store some token (`wmdToken`):

```
contract BasicVault {

    IERC20 public immutable wmdToken;
    mapping(address => uint) public balances;

    event Deposited(address indexed from, uint amount);
    event Withdrawal(address indexed from, uint amount);

    constructor(IERC20 wmdToken_){
        wmdToken = wmdToken_;
    }

    function deposit(uint amount) external {
        balances[msg.sender] += amount;
        bool success = wmdToken.transferFrom(msg.sender, address(this),
amount);
        require(success, "Deposit failed!");
        emit Deposited(msg.sender, amount);
    }

    function withdraw(uint amount) external {
        balances[msg.sender] -= amount;
        bool success = wmdToken.transfer(msg.sender, amount);
        require(success, "Withdrawal failed!");
        emit Withdrawal(msg.sender, amount);
    }
}
```

We have a test function to ensure that a user is unable to withdraw tokens in excess of his deposit, like so:

```
function testUserCannotWithdrawExcessOfDeposit() public {
    vm.prank(user);
    vm.expectRevert(stdError.arithmeticError);
    vault.withdraw(userTokens + 100*10**18);
}
```

1. User has tokens of amount `userTokens` deposited in a Vault contract.
2. User attempts to withdraw tokens of amount in excess of his deposits.
3. This leads to an underflow error, resulting from: `balances[msg.sender] -= amount;` as it would evaluate into a negative value.

To catch the error "Arithmetic over/underflow", we insert `vm.expectRevert(stdError.arithmeticError)` just before the function call that is expected to result in an underflow.

divisionError

Signature

```
stdError.divisionError
```

Description

The internal Solidity error when a division fails, e.g. division by zero.

enumConversionError

Signature

```
stdError.enumConversionError
```

Description

The internal Solidity error when trying to convert a number to a variant of an enum, if the number is larger than the number of variants in the enum (counting from 0).

encodeStorageError

Signature

```
stdError.encodeStorageError
```

Description

The internal Solidity error when trying to access data in storage that is corrupted. Data cannot be corrupted unless assembly had been used.

popError

Signature

```
stdError.popError
```

Description

The internal Solidity error when trying to pop an element off of an empty array.

`indexOOBError`

Signature

```
stdError.indexOOBError
```

Description

The internal Solidity error when trying to access an element of an array that is out of bounds.

Will not work for empty arrays in external contracts. For those, use `expectRevert` without any arguments.

memOverflowError

Signature

```
stdError.memOverflowError
```

Description

The internal Solidity error when trying to allocate a dynamic memory array with more than 2^64-1 items.

zeroVarError

Signature

```
stdError.zeroVarError
```

Description

The internal Solidity error when trying to call a function via a function pointer that has not been initialized.

Std Storage

Std Storage is a library that makes manipulating storage easy.

To use Std Storage, import the following in your test contract:

```
import {stdStorage, StdStorage} from "forge-std/Test.sol";
```

Add the following line in your test contract:

```
using stdStorage for StdStorage;
```

Then, access Std Storage via the `stdstore` instance.

Functions

Query functions:

- `target` : Set the address of the target contract
- `sig` : Set the 4-byte selector of the function to static call
- `with_key` : Pass an argument to the function (can be used multiple times)
- `depth` : Set the position of the value in the `tuple` (e.g. inside a `struct`)

Terminator functions:

- `find` : Return the slot number
- `checked_write` : Set the data to be written to the storage slot(s)
- `read_<type>` : Read the value from the storage slot as `<type>`

Example

`playerToCharacter` tracks info about players' characters.

```
// MetaRPG.sol

struct Character {
    string name;
    uint256 level;
}

mapping (address => Character) public playerToCharacter;
```

Let's say we want to set the level of our character to 120.

```
// MetaRPG.t.sol

stdstore
    .target(address(metaRpg))
    .sig("playerToCharacter(address)")
    .with_key(address(this))
    .depth(1)
    .checked_write(120);
```

Limitations

- Accessing packed slots is not supported

Known issues

- Slot(s) may not be found if the `tuple` contains types shorter than 32 bytes

target

Signature

```
function target(StdStorage storage self, address _target) internal returns
(StdStorage storage);
```

Description

Sets the address of the contract.

Default value: `address(0)`

sig

Signature

```
function sig(StdStorage storage self, bytes4 _sig) internal returns (StdStorage
storage);
```

```
function sig(StdStorage storage self, string memory _sig) internal returns
(StdStorage storage);
```

Description

Sets the 4-byte selector of the function to static call.

Default value: `hex"00000000"`

Examples

```
uint256 slot = stdstore
    .target(addr)
    .sig(addr.fun.selector)
    .with_key(1)
    .find();
```

```
uint256 slot = stdstore
    .target(addr)
    .sig("fun(uint256)")
    .with_key(1)
    .find();
```

with_key

Signature

```
function with_key(StdStorage storage self, address who) internal returns
(StdStorage storage);
```

```
function with_key(StdStorage storage self, uint256 amt) internal returns
(StdStorage storage);
```

```
function with_key(StdStorage storage self, bytes32 key) internal returns
(StdStorage storage);
```

Description

Passes an argument to the function.

Can be used multiple times to pass multiple arguments. The order matters.

Examples

```
uint256 slot = stdstore
    .target(addr)
    .sig("fun(uint256,address)")
    .with_key(1)
    .with_key(address(this))
    .find();
```

depth

Signature

```
function depth(StdStorage storage self, uint256 _depth) internal returns
(StdStorage storage);
```

Description

Sets the position of the value in the `tuple` (e.g. inside a `struct`).

Default value: `uint256(0)`

checked_write

Signature

```
function checked_write(StdStorage storage self, address who) internal;

function checked_write(StdStorage storage self, uint256 amt) internal;

function checked_write(StdStorage storage self, bool write) internal;

function checked_write(StdStorage storage self, bytes32 set) internal;

function checked_write_int(StdStorage storage self, int256 val) internal;
```

Description

Sets the data to be written to the storage slot(s).

Reverts with a message if unsuccessful.

`find`

Signature

```
function find(StdStorage storage self) internal returns (uint256);
```

Description

Finds an arbitrary storage slot given `target`, `sig`, `with_key`(s), and `depth`.

Reverts with a message if unsuccessful.

read

Signature

```
function read_bytes32(StdStorage storage self) internal returns (bytes32);

function read_bool(StdStorage storage self) internal returns (bool);

function read_address(StdStorage storage self) internal returns (address);

function read_uint(StdStorage storage self) internal returns (uint256);

function read_int(StdStorage storage self) internal returns (int256);
```

Description

Reads the value from the storage slot as `bytes32`, `bool`, `address`, `uint256`, or `int256`.

Reverts with a message if unsuccessful.

Std Math

- `abs`
- `delta`
- `percentDelta`

abs

Signature

```
function abs(int256 a) internal pure returns (uint256)
```

Description

Returns the absolute value of a number.

Example

```
uint256 ten = stdMath.abs(-10);
```

delta

Signature

```
function delta(uint256 a, uint256 b) internal pure returns (uint256)
```

```
function delta(int256 a, int256 b) internal pure returns (uint256)
```

Description

Returns the difference between two numbers in absolute value.

Example

```
uint256 four = stdMath.delta(-1, 3);
```

percentDelta

Signature

```
function percentDelta(uint256 a, uint256 b) internal pure returns (uint256)
```

```
function percentDelta(int256 a, int256 b) internal pure returns (uint256)
```

Description

Returns the difference between two numbers in percentage, where `1e18` is 100%. More precisely, `percentDelta(a, b)` computes `abs((a-b) / b) * 1e18`.

Example

```
uint256 percent150 = stdMath.percentDelta(uint256(125), 50);
uint256 percent60 = stdMath.percentDelta(uint256(50), 125);
```

Script Utils

- `computeCreateAddress`
- `deriveRememberKey`

computeCreateAddress

Signature

```
function computeCreateAddress(address deployer, uint256 nonce) internal pure
returns (address)
```

Description

Compute the address a contract will be deployed at for a given deployer address and nonce. Useful to precalculate the address a contract **will** be deployed at.

Example

```
address governanceAddress =
computeCreateAddress(0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266, 1);

// this contract requires a governance contract which hasn't been deployed yet
Contract contract = new Contract(governanceAddress);
// now we deploy it
Governance governance = new Governance(contract);

// assuming `contract` has a `governance()` accessor
assertEq(governanceAddress, address(governance)); // [PASS]
```

deriveRememberKey

Signature

```
function deriveRememberKey(string memory mnemonic, uint32 index) internal
returns (address who, uint256 privateKey)
```

Description

Derive a private key from a mnemonic and also store it in forge's local wallet. Returns the address and private key.

Example

Get a private key and address from the test mnemonic at path `m/44'/60'/0'/0/0`. Use them to sign some data and start broadcasting transactions:

```
string memory mnemonic = "test test test test test test test test test test
test junk";

(address deployer, uint256 privateKey) = deriveRememberKey(mnemonic, 0);

bytes32 hash = keccak256("Signed by deployer");
(uint8 v, bytes32 r, bytes32 s) = vm.sign(privateKey, hash);

vm.startBroadcast(deployer);
...
vm.stopBroadcast();
```

Get an address from the test mnemonic at path `m/44'/60'/0'/0/0` to start broadcasting transactions:

```
string memory mnemonic = "test test test test test test test test test test
test junk";

(address deployer, ) = deriveRememberKey(mnemonic, 0);

vm.startBroadcast(deployer);
...
vm.stopBroadcast();
```

SEE ALSO

Cheatcodes:

- deriveKey
- rememberKey

Console Logging

- Similar to Hardhat's console functions.
- You can use it in calls and transactions. It works with view functions, but not in pure ones.
- It always works, regardless of the call or transaction failing or being successful.
- To use it you need import it:
  - `import "forge-std/console.sol";`
- You can call console.log with up to 4 parameters in any order of following types:
  - `uint`
  - `string`
  - `bool`
  - `address`
- There's also the single parameter API for the types above, and additionally bytes, bytes1... up to bytes32:
  - `console.logInt(int i)`
  - `console.logUint(uint i)`
  - `console.logString(string memory s)`
  - `console.logBool(bool b)`
  - `console.logAddress(address a)`
  - `console.logBytes(bytes memory b)`
  - `console.logBytes1(bytes1 b)`
  - `console.logBytes2(bytes2 b)`
  - ...
  - `console.logBytes32(bytes32 b)`
- console.log implements the same formatting options that can be found in Hardhat's console.log.
  - Example: `console.log("Changing owner from %s to %s", currentOwner, newOwner)`
- console.log is implemented in standard Solidity and it is compatible Anvil and Hardhat Networks.
- console.log calls can run in other networks, like mainnet, kovan, ropsten, etc. They do nothing in those networks, but do spend a minimal amount of gas.

`console.log(format[,...args])`

The `console.log()` method prints a formatted string using the first argument as a printf-like format string which can contain zero or more format specifiers. Each specifier is replaced with the converted value from the corresponding argument. Supported specifiers are:

- `%s` : String will be used to convert all values to a human-readable string. `uint256`, `int256` and `bytes` values are converted to their `0x` hex encoded values.
- `%d` : Number will be used to convert all values to a human-readable string. This is identical to `%s`.
- `%i` : Works the same way as `%d`.
- `%e` : The exponential representation of a number. For `uint256` and `int256` types.
- `%x` : The hexadecimal representation of a number. For `uint256` and `int256` types.
- `%o` : Object. A string representation of an object with generic JavaScript-styled object formatting. For solidity types, this basically surround the string representation of the value in single-quotes.
- `%%` : single percent sign ('%'). This does not consume an argument.
- Returns: `<string>` The formatted string

If a specifier does not have a corresponding argument, it is not replaced:

```
console.log("%s:%s", "foo");
// Returns: "foo:%s"
```

Values that are not part of the format string are formatted using as a human-readable string representation.

If there are more arguments passed to the console.log() method than the number of specifiers, the extra arguments are concatenated to the returned string, separated by spaces:

```
console.log("%s:%s", "foo", "bar", "baz");
// Returns: "foo:bar baz"
```

If only one argument is passed to console.log(), it is returned as it is without any formatting:

```
console.log("%% %s");
// Returns: "%% %s"
```

The String format specifier (`%s`) should be used in most cases unless specific functionality is needed from other format specifiers.

DSTest Reference

Dappsys Test (DSTest for short) provides basic logging and assertion functionality. It is included in the Forge Standard Library.

To get access to the functions, import `forge-std/Test.sol` and inherit from `Test` in your test contract:

```
import "forge-std/Test.sol";

contract ContractTest is Test {
    // ... tests ...
}
```

Logging

This is a complete overview of all the available logging events. For detailed descriptions and example usage, see below.

```
event log                    (string);
event logs                   (bytes);

event log_address            (address);
event log_bytes32            (bytes32);
event log_int                (int);
event log_uint               (uint);
event log_bytes              (bytes);
event log_string             (string);

event log_named_address      (string key, address val);
event log_named_bytes32      (string key, bytes32 val);
event log_named_decimal_int  (string key, int val, uint decimals);
event log_named_decimal_uint (string key, uint val, uint decimals);
event log_named_int          (string key, int val);
event log_named_uint         (string key, uint val);
event log_named_bytes        (string key, bytes val);
event log_named_string       (string key, string val);
```

Logging events

This section documents all events for logging and provides usage examples.

log

```
event log(string);
```

Example

```
emit log("here");
// here
```

---

logs

```
event logs(bytes);
```

Example

```
emit logs(bytes("abcd"));
// 0x6162636400000000000000000000000000000000000000000000000000000000
```

---

log_<type>

```
event log_<type>(<type>);
```

Where `<type>` can be `address`, `bytes32`, `int`, `uint`, `bytes`, `string`

Example

```
uint256 amount = 1 ether;
emit log_uint(amount);
// 1000000000000000000
```

---

log_named_<type>

```
event log_named_<type>(string key, <type> val);
```

Where `<type>` can be `address`, `bytes32`, `int`, `uint`, `bytes`, `string`

Example

```
uint256 amount = 1 ether;
emit log_named_uint("Amount", amount);
// amount: 1000000000000000000
```

---

log_named_decimal_<type>

```
event log_named_decimal_<type>(string key, <type> val, uint decimals);
```

Where `<type>` can be `int`, `uint`

Example

```
uint256 amount = 1 ether;
emit log_named_decimal_uint("Amount", amount, 18);
// amount: 1.000000000000000000
```

Asserting

This is a complete overview of all the available assertion functions. For detailed descriptions and example usage, see below.

```
// Assert the `condition` is true
function assertTrue(bool condition) internal;
function assertTrue(bool condition, string memory err) internal;

// Assert `a` is equal to `b`
function assertEq(address a, address b) internal;
function assertEq(address a, address b, string memory err) internal;
function assertEq(bytes32 a, bytes32 b) internal;
function assertEq(bytes32 a, bytes32 b, string memory err) internal;
function assertEq(int a, int b) internal;
function assertEq(int a, int b, string memory err) internal;
function assertEq(uint a, uint b) internal;
function assertEq(uint a, uint b, string memory err) internal;
function assertEqDecimal(int a, int b, uint decimals) internal;
function assertEqDecimal(int a, int b, uint decimals, string memory err)
internal;
function assertEqDecimal(uint a, uint b, uint decimals) internal;
function assertEqDecimal(uint a, uint b, uint decimals, string memory err)
internal;
function assertEq(string memory a, string memory b) internal;
function assertEq(string memory a, string memory b, string memory err)
internal;
function assertEq32(bytes32 a, bytes32 b) internal;
function assertEq32(bytes32 a, bytes32 b, string memory err) internal;
function assertEq0(bytes memory a, bytes memory b) internal;
function assertEq0(bytes memory a, bytes memory b, string memory err) internal;

// Assert  `a` is greater than `b`
function assertGt(uint a, uint b) internal;
function assertGt(uint a, uint b, string memory err) internal;
function assertGt(int a, int b) internal;
function assertGt(int a, int b, string memory err) internal;
function assertGtDecimal(int a, int b, uint decimals) internal;
function assertGtDecimal(int a, int b, uint decimals, string memory err)
internal;
function assertGtDecimal(uint a, uint b, uint decimals) internal;
function assertGtDecimal(uint a, uint b, uint decimals, string memory err)
internal;

// Assert  `a` is greater than or equal to `b`
function assertGe(uint a, uint b) internal;
function assertGe(uint a, uint b, string memory err) internal;
function assertGe(int a, int b) internal;
function assertGe(int a, int b, string memory err) internal;
function assertGeDecimal(int a, int b, uint decimals) internal;
function assertGeDecimal(int a, int b, uint decimals, string memory err)
internal;
function assertGeDecimal(uint a, uint b, uint decimals) internal;
function assertGeDecimal(uint a, uint b, uint decimals, string memory err)
internal;

// Assert  `a` is lesser than `b`
function assertLt(uint a, uint b) internal;
function assertLt(uint a, uint b, string memory err) internal;
function assertLt(int a, int b) internal;
function assertLt(int a, int b, string memory err) internal;
function assertLtDecimal(int a, int b, uint decimals) internal;
```

```
function assertLtDecimal(int a, int b, uint decimals, string memory err)
internal;
function assertLtDecimal(uint a, uint b, uint decimals) internal;
function assertLtDecimal(uint a, uint b, uint decimals, string memory err)
internal;

// Assert  `a` is lesser than or equal to `b`
function assertLe(uint a, uint b) internal;
function assertLe(uint a, uint b, string memory err) internal;
function assertLe(int a, int b) internal;
function assertLe(int a, int b, string memory err) internal;
function assertLeDecimal(int a, int b, uint decimals) internal;
function assertLeDecimal(int a, int b, uint decimals, string memory err)
internal;
function assertLeDecimal(uint a, uint b, uint decimals) internal;
function assertLeDecimal(uint a, uint b, uint decimals, string memory err)
internal;

// Asserts `a` is approximately equal to `b` with delta in absolute value.
function assertApproxEqAbs(uint256 a, uint256 b, uint256 maxDelta) internal;
function assertApproxEqAbs(uint256 a, uint256 b, uint256 maxDelta, string
memory err) internal;

// Asserts `a` is approximately equal to `b` with delta in percentage, where
`1e18` is 100%.
function assertApproxEqRel(uint256 a, uint256 b, uint256 maxPercentDelta)
internal;
function assertApproxEqRel(uint256 a, uint256 b, uint256 maxPercentDelta,
string memory err) internal;
```

Assertion functions

This section documents all functions for asserting and provides usage examples.

`assertTrue`

```
function assertTrue(bool condition) internal;
```

Asserts the `condition` is true.

Example

```
bool success = contract.fun();
assertTrue(success);
```

---

assertEq

```
function assertEq(<type> a, <type> b) internal;
```

Where `<type>` can be `address`, `bytes32`, `int`, `uint`

Asserts `a` is equal to `b`.

Example

```
uint256 a = 1 ether;
uint256 b = 1e18 wei;
assertEq(a, b);
```

---

assertEqDecimal

```
function assertEqDecimal(<type> a, <type> b, uint decimals) internal;
```

Where `<type>` can be `int`, `uint`

Asserts `a` is equal to `b`.

Example

```
uint256 a = 1 ether;
uint256 b = 1e18 wei;
assertEqDecimal(a, b, 18);
```

---

assertEq32

```
function assertEq32(bytes32 a, bytes32 b) internal;
```

Asserts `a` is equal to `b`.

Example

```
assertEq(bytes32("abcd"),
0x6162636400000000000000000000000000000000000000000000000000000000);
```

assertEq0

```
function assertEq0(bytes a, bytes b) internal;
```

Asserts `a` is equal to `b`.

Example

```
string memory name1 = "Alice";
string memory name2 = "Bob";
assertEq0(bytes(name1), bytes(name2)); // [FAIL]
```

---

assertGt

```
function assertGt(<type> a, <type> b) internal;
```

Where `<type>` can be `int`, `uint`

Asserts `a` is greater than `b`.

Example

```
uint256 a = 2 ether;
uint256 b = 1e18 wei;
assertGt(a, b);
```

---

assertGtDecimal

```
function assertGtDecimal(<type> a, <type> b, uint decimals) internal;
```

Where `<type>` can be `int`, `uint`

Asserts `a` is greater than `b`.

Example

```
uint256 a = 2 ether;
uint256 b = 1e18 wei;
assertGtDecimal(a, b, 18);
```

---

assertGe

```
function assertGe(<type> a, <type> b) internal;
```

Where `<type>` can be `int`, `uint`

Asserts `a` is greater than or equal to `b`.

Example

```
uint256 a = 1 ether;
uint256 b = 1e18 wei;
assertGe(a, b);
```

---

assertGeDecimal

```
function assertGeDecimal(<type> a, <type> b, uint decimals) internal;
```

Where `<type>` can be `int`, `uint`

Asserts `a` is greater than or equal to `b`.

Example

```
uint256 a = 1 ether;
uint256 b = 1e18 wei;
assertGeDecimal(a, b, 18);
```

---

assertLt

```
function assertLt(<type> a, <type> b) internal;
```

Where `<type>` can be `int`, `uint`

Asserts `a` is lesser than `b`.

Example

```
uint256 a = 1 ether;
uint256 b = 2e18 wei;
assertLt(a, b);
```

---

assertLtDecimal

```
function assertLtDecimal(<type> a, <type> b, uint decimals) internal;
```

Where `<type>` can be `int`, `uint`

Asserts `a` is lesser than `b`.

Example

```
uint256 a = 1 ether;
uint256 b = 2e18 wei;
assertLtDecimal(a, b, 18);
```

---

assertLe

```
function assertLe(<type> a, <type> b) internal;
```

Where `<type>` can be `int`, `uint`

Asserts `a` is lesser than or equal to `b`.

Example

```
uint256 a = 1 ether;
uint256 b = 1e18 wei;
assertLe(a, b);
```

---

assertLeDecimal

```
function assertLeDecimal(<type> a, <type> b, uint decimals) internal;
```

Where `<type>` can be `int`, `uint`

Asserts `a` is lesser than or equal to `b`.

Example

```
uint256 a = 1 ether;
uint256 b = 1e18 wei;
assertLeDecimal(a, b, 18);
```

---

assertApproxEqAbs

```
function assertApproxEqAbs(<type> a, <type> b, uint256 maxDelta) internal;
```

Where `<type>` can be `int`, `uint`

Asserts `a` is approximately equal to `b` with delta in absolute value.

Example

```
function testFail () external {
    uint256 a = 100;
    uint256 b = 200;

    assertApproxEqAbs(a, b, 90);
}
```

---

assertApproxEqRel

```
function assertApproxEqRel(<type> a, <type> b, uint256 maxPercentDelta)
internal;
```

Where `<type>` can be `int`, `uint`

Asserts `a` is approximately equal to `b` with delta in percentage, where `1e18` is 100%.

Example

```
function testFail () external {
    uint256 a = 100;
    uint256 b = 200;
    assertApproxEqRel(a, b, 0.4e18);
}
```

ℹ️ Information

You can pass a custom error message to the above functions by providing an additional parameter `string err`.

Miscellaneous

- Struct encoding
- Precompile Registry

Struct Encoding

Structs are user defined types that can group several variables:

```
struct MyStruct {
    address addr;
    uint256 amount;
}
```

Only the new ABI coder v2 can encode and decode arbitrarily nested arrays and structs. Since Solidity 0.8.0 it is activated by default, prior to that it needs to be activated via `pragma experimental ABIEncoderV2`.

Solidity structs map to the ABI type "tuple". For more information on how Solidity types map to ABI types see Mapping Solidity to ABI types in the Solidity documentation.

Structs are therefore encoded and decoded as tuples. So the struct we defined above, `MyStruct`, maps to the tuple `(address,uint256)` in terms of the ABI.

Let's see how this works in a contract:

```
pragma solidity =0.8.15;


contract Test {
    struct MyStruct {
        address addr;
        uint256 amount;
    }
    function f(MyStruct memory t) public pure {}
}
```

The ABI of the `f` function in this contract is:

```
{
    "inputs": [
        {
            "components": [
                {
                    "internalType": "address",
                    "name": "addr",
                    "type": "address"
                },
                {
                    "internalType": "uint256",
                    "name": "amount",
                    "type": "uint256"
                }
            ],
            "internalType": "struct Test.MyStruct",
            "name": "t",
            "type": "tuple"
        }
    ],
    "name": "f",
    "outputs": [],
    "stateMutability": "pure",
    "type": "function"
}
```

which reads: The function `f` takes 1 input of type `tuple` with two components of type `address` and `uint256`.

Nested Struct Encoding: Here's a more complex example with nested structs:

```
pragma solidity 0.8.21;

contract Test {
    struct nestedStruct {
        address addr;
        uint256 amount;
    }

    struct MyStruct {
        string nestedStructName;
        uint256 nestedStructCount;
        nestedStruct _nestedStruct;
    }

    function f(MyStruct memory t) public pure {}
}
```

The ABI of the `f` function in this contract is:

```
{
    "inputs": [
        {
            "name": "t",
            "type": "tuple",
            "internalType": "struct Test.MyStruct",
            "components": [
                {
                    "name": "nestedStructName",
                    "type": "string",
                    "internalType": "string"
                },
                {
                    "name": "nestedStructCount",
                    "type": "uint256",
                    "internalType": "uint256"
                },
                {
                    "name": "_nestedStruct",
                    "type": "tuple",
                    "internalType": "struct Test.nestedStruct",
                    "components": [
                        {
                            "name": "addr",
                            "type": "address",
                            "internalType": "address"
                        },
                        {
                            "name": "amount",
                            "type": "uint256",
                            "internalType": "uint256"
                        }
                    ]
                }
            ]
        }
    ],
    "name": "f",
    "outputs": [],
    "stateMutability": "pure",
    "type": "function"
}
```

which reads: The function `f` takes 1 input of type tuple with three components: a string, a uint256, and another tuple representing the nested struct with components addr of type address and amount of type uint256.

To encode `MyStruct` to pass it as a parameter to the function `f`:

```
cast abi-encode "f((string,uint256,(address,uint256)))" "(example,1,(0x...,1))"
```

To deploy a contract accepting `MyStruct` as an argument:

```
forge create src/Test.sol:Test --constructor-args "(example,1,(0x...,1))"
```

Precompile Registry

Precompiles are special contracts at fixed addresses that are included within the EVM. In addition to common precompiles included with other EVM environments, Foundry includes a few precompiles for environment mutation, logging data, and contract deployment.

Note that, while some chains like Optimism have bytecode deployed at a predetermined address, making them 'pre-deploys', we treat them as precompiles within the context of Foundry.

Registry

Chain ID	Address	Name
ALL	`0x01`	ECRecover
ALL	`0x02`	SHA-256
ALL	`0x03`	RIPEMD-160
ALL	`0x04`	Identity
ALL	`0x05`	ModExp
ALL	`0x06`	ECAdd
ALL	`0x07`	ECMul
ALL	`0x08`	ECPairing
ALL	`0x09`	Blake2F
10, 420	`0x4200000000000000000000000000000000000016`	L2ToL1MessagePasser
10, 420	`0x4200000000000000000000000000000000000002`	DeployerWhitelist
10, 420	`0xDeadDeAddeAddEAddeadDEaDDEAdDeaDDeAD0000`	LegacyERC20ETH
10, 420	`0x4200000000000000000000000000000000000006`	WETH9
10, 420	`0x4200000000000000000000000000000000000007`	L2CrossDomainMesse
10, 420	`0x4200000000000000000000000000000000000010`	L2StandardBridge
10, 420	`0x4200000000000000000000000000000000000011`	SequencerFeeVault
10, 420	`0x4200000000000000000000000000000000000012`	OptimismMintableERC
10, 420	`0x4200000000000000000000000000000000000013`	L1BlockNumber
10, 420	`0x420000000000000000000000000000000000000F`	GasPriceOracle
10, 420	`0x4200000000000000000000000000000000000015`	L1Block
10, 420	`0x4200000000000000000000000000000000000042`	GovernanceToken

Chain ID	Address	Name
10, 420	0x4200000000000000000000000000000000000000	LegacyMessagePasser
10, 420	0x4200000000000000000000000000000000000014	L2ERC721Bridge
10, 420	0x4200000000000000000000000000000000000017	OptimismMintableERC
10, 420	0x4200000000000000000000000000000000000018	ProxyAdmin
42161, 421613	0x0000000000000000000000000000000000000064	ArbSys
42161, 421613	0x000000000000000000000000000000000000006E	ArbRetryableTx
42161, 421613	0x000000000000000000000000000000000000006C	ArbGasInfo
42161, 421613	0x0000000000000000000000000000000000000066	ArbAddressTable
42161, 421613	0x000000000000000000000000000000000000006F	ArbStatistics
42161, 421613	0x00000000000000000000000000000000000000C8	NodeInterface
42161, 421613	0x0000000000000000000000000000000000000067	ArbBLS
42161, 421613	0x0000000000000000000000000000000000000065	ArbInfo
42161, 421613	0x000000000000000000000000000000000000006D	ArbAggregator
42161, 421613	0x0000000000000000000000000000000000000068	ArbFunctionTable
433114, 43113	0x0200000000000000000000000000000000000000	ContractDeployerAllov
433114, 43113	0x0200000000000000000000000000000000000001	ContractNativeMinterA
433114, 43113	0x0200000000000000000000000000000000000002	TxAllowListAddress
433114, 43113	0x0200000000000000000000000000000000000003	FeeConfigManagerAdc
ALL	0x4e59b44847b379578588920cA78FbF26c0B4956C	(Foundry) Create2Depl
ALL	0x7109709ECfa91a80626fF3989D68f67F5b1DD12D	(Foundry) VM
ALL	0x000000000000000000636F6e736F6c652e6c6f67	(Foundry) Console

Reserved Ranges

Some chains also include reserved ranges for precompile contracts.

Chain ID	Start	S
ALL	`0x00`	`0xff`
433114, 43113	`0x0100000000000000000000000000000000000000`	`0x0100000000000000000`
433114, 43113	`0x0200000000000000000000000000000000000000`	`0x0200000000000000000`
433114, 43113	`0x0300000000000000000000000000000000000000`	`0x0300000000000000000`